Lauri af Heurlin

Authorization Certificate based Access Control in Embedded Environments

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 11.05.2015

Thesis supervisor:

Prof. Valeriy Vyatkin

Thesis advisor:

Lic.Sc. (Tech.) Yki Kortesniemi

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Lauri af Heurlin

Title: Authorization Certificate based Access Control in Embedded Environments

Date: 11.05.2015 Language: English Number of pages: 7+64

Department of Automation and Systems Technology

Professorship: Information and Computer Systems in Automation Code: AS220-3

Supervisor: Prof. Valeriy Vyatkin

Advisor: Lic.Sc. (Tech.) Yki Kortesniemi

In this thesis, I study the problem of implementing a distributed access control solution on an embedded device. As an example I will have a parking service, which employs cheap embedded devices at every parking lot to guard them against unauthorized usage.

I implemented a prototype using SPKI Authorization certificate based access control system on an embedded platform. It uses a smartphone as the Client device and Bluetooth Low Energy as the wireless communication link between the Client and the embedded platform.

To better evaluate the performance achievable on embedded devices, the system was implemented on two different platforms. Certificate Chain Reduction and compressed public keys were also studied as ways to improve the system's performance.

The results show that in our use case, the faster platform can evaluate access permissions within 1.3 s from Client device starting device discovery and the slower one within 3.2 s. Chain Reduction was found to give significant benefits, decreasing time required for evaluation by over 40%, to 0.8 s on faster platform, while using compressed public keys only reduced time required by 10% on it. However, both numbers are dependent on the embedded and wireless platforms used. The faster platform has good performance in our use case and the cheapness of embedded devices enables their liberal use in distributed environments.

In conclusion, I find that SPKI Authorization certificates work well as a basis for a distributed access control system, and that modern embedded devices are fast enough to provide more than sufficient service level even if the cryptography is implemented entirely with software. One potential application area for this technology are Internet-of-Things devices, which would benefit greatly from a distributed access control system.

Keywords: Elliptic curve cryptography, Embedded systems, SPKI Certificates, Authentication Certificates, Access Control, Distributed systems

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Lauri af Heurlin

Työn nimi: Valtuussertifikaattipohjainen pääsynhallinta sulautetuissa järjestelmissä

Päivämäärä: 11.05.2015 Kieli: Englanti Sivumäärä: 7+64

Automaatio- ja Systeemitekniikan laitos

Professuuri: Automaation tietotekniikka ja -järjestelmät Koodi: AS220-3

Valvoja: Prof. Valeriy Vyatkin

Ohjaaja: Tekn. Lis Yki Kortesniemi

Tässä diplomityössä tutkin hajautetun pääsynhallinnan toteuttamista sulautetuissa järjestelmissä. Esimerkkinä käytän pysäköintipalvelua, joka hyödyntää edullisia sulautettuja laitteita palvelun hallitsemien pysäköintialueiden luvattoman käytön estämiseen.
Mittauksia varten toteutin prototyypin SPKI-valtuutussertifikaatteihin perustuvasta pääsynhallintajärjestelmästä käyttäen sulautettua alustaa. Järjestelmän asiakaslaitteena on älypuhelin, ja langaton tiedonsiirtoyhteys asiakaslaitteen ja sulautetun alustan välillä on toteutettu Bluetooth Low Energyllä.
Järjestelmä toteuttiin kahta eri alustaa käyttäen, jotta voitiin paremmin tutkia sulautetuilla laitteilla saavutettavissa olevaa suorituskykyä. Ketjureduktiota ja julkisten avainten pakkausta tutkittiin mahdollisina keinoina parantaa järjestelmän suorituskykyä.
Tulokset osoittavat, että esimerkissämme nopeampi alusta kykenee tarkastamaan pääsyoikeudet 1.3 sekunnin kuluessa siitä, kun asiakaslaite aloittaa Bluetooth Low Energy -laitteiden löytämisen. Hitaampi alusta kykenee samaan 3.2 sekunnissa. Ketjureduktiolla saavutettiin merkittäviä etuja: tarkastukseen kulunut aika lyheni 40%, 0.8 sekuntiin nopeammalla alustalla.
Pakattujen julkisten avainten käyttö taas lyhensi aikaa vain kymmenellä prosentilla. Molemmat tulokset kuitenkin riippuvat käytettävissä olevista sulautetuista- ja tiedonsiirtoalustoista.
Yhteenvetona totean, että SPKI-valtuutussertifikaatit sopivat hyvin hajautetun pääsynhallintajärjestelmän perustaksi, ja nykyaikaisten sulautettujen laitteiden nopeus mahdollistaa riittävän palvelutason myös silloin, kun salaustekniikka toteutetaan kokonaan ohjelmistolla. Tämän teknologian yksi mahdollinen sovelluskohde ovat "esineiden internetiin" kytkeytyvät laitteet, jotka hyötyisivät suuresti hajautetusta pääsynhallinnasta.

Avainsanat: Sulautetut järjestelmät, Elliptisten käyrien kryptosysteemit, SPKI sertifikaatit, Pääsynhallinta, Hajautetut järjestelmät

Preface

I want to thank my supervisor, Professor Valeriy Vyatkin, and my advisor Yki Kortesniemi for dedicated and detailed advisory. I thank HIIT for this interesting thesis project and for creating a stimulating work environment.

I'm also grateful for the members of ELL-i Open Source Cooperative, who were a tremendous help with solving issues related to the embedded platform.

Finally, I give my warmest thanks to my wife Johanna, for her support and patience during this project.

Otaniemi, 11.05.2015

Lauri af Heurlin

Contents

Symbols and abbreviations

Abbreviations

BLE	Bluetooth Low Energy
BT	Bluetooth
CPU	Central Processing Unit
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
DMIPS	Dhrystone Millions of Instructions Per Second
GAP	Generic Access Profile
GATT	Generic Attribute Profile
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
RAM	Random Access Memory
SEC	Standards for Efficient Cryptography
SHA-2	Secure Hashing Algorithm 2
SPI	Serial Peripheral Interface
SPIRE	Smart Parking fo Intelligent Real Estate
SPKI	Simple Public Key Infrastructure
RNG	Random Number Generator
RSA	Rivest, Shamir & Adleman (public key cryptosystem)
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

1 Introduction

This thesis studies the problem of implementing an access control solution for a parking service and the performance attainable by using embedded systems in its implementation.

In the introduction section, I first go through the use case for the parking service and detail the business requirements arising from it. Next, I present the scope of the system implementation, and introduce the research questions, which concentrate on the performance of the system. Finally, the structure of the thesis is detailed.

1.1 Use case

Smart Parking for Intelligent Real Estate (SPIRE) [11] project has created a flexible parking service called EnterLot. It provides users with information about available parking spaces and their costs, guides the users to the chosen parking lot with turn-by-turn navigation, and also provides indoor guidance to the final destination inside the building.

In order to provide a complete solution to parking, EnterLot also requires an access control solution for the parking lots. In this thesis, a proof-of-concept keyless, wireless access control system usable by a smartphone was implemented and evaluated. Main components of the system are the Client and the Barrier, as shown in the Figure 1

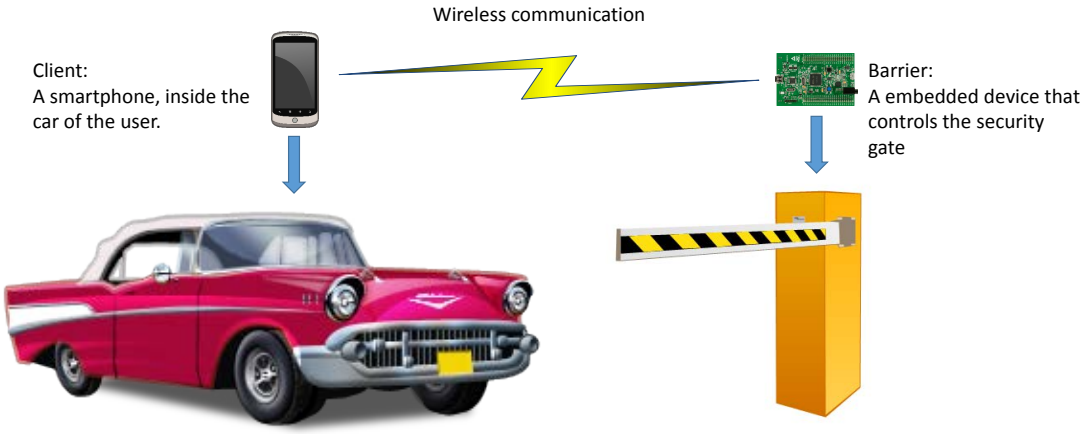


Figure 1: Overview of the components of the Access control system

Our use case starts with a driver driving to the entrance of the parking lot, for which they have a valid monthly parking permit.

The entrance to the parking lot is restricted by a gate and the embedded system controlling the gate is called the Barrier.

At the entrance the software running on driver's smartphone (henceforth referred as the Client) will automatically attempt to wirelessly negotiate with the Barrier access to the parking lot, without requiring any user intervention in our use case as the software already knows what parking lot the driver wishes to park at. If the driver has a valid parking permit, the Barrier will open the gate, allowing driver to proceed to the area. If not, the Client will show an error message to the driver, explaining the reason for the access being denied.

The process, starting with the Client initiating communications with Barrier, and ending with the Barrier having decided to grant access to the parking lot, will henceforth be called the *transaction*. From the driver's point of view, the transaction starts when he or she drives the car to the gate and ends when the gate starts to rise.

As parking lots are often situated in areas without constant network connectivity, our use case is built on the assumption that the system does not require a network connection during the transaction, and certificates will be used to enable offline use.

Our business requirements can be derived from the EnterLot system and the use case above, and in condensed form they are as follows:

Business Requirement 1. *System will be distributed: system has to support multiple parking areas, each with its own Barrier.*

Business Requirement 2. *The Barrier must work without requiring constant network connection.*

Business Requirement 3. *The Barrier will run on an embedded platform.*

Business Requirement 4. *The Client device will be a smartphone.*

Business Requirement 5. *The user shouldn't be required to open the window and place the phone near the Barrier: in other words, a range of several meters is required.*

Business Requirement 6. *To ensure a smooth user experience, the system has to respond in timely manner: the whole transaction, from requesting access to the parking lot to the access being granted, should take at most 1.5 seconds, preferably less than 1 second.*

1.2 Scope

The goal of this project was to study the problem of implementing a distributed access control solution on embedded platform. I therefore implemented a prototype where the Barrier is an embedded device, the Client is a smartphone and SPKI authorization certificates were used as the access control mechanic.

Main focus of the project was on establishing baseline performance for the embedded platform, so particular attention was given to those parts of the system. For

example, the Barrier was implemented with two separate embedded platforms so that the different performance aspects could be more easily evaluated.

However, as modern smartphones are significantly faster than embedded devices, I chose not to implement a fully functional Client software with the assumption that it will not affect the performance measurements of the embedded system to any significant degree. In the Analysis section (Chapter 6) of this thesis I will show that smartphone does indeed have ample time to do any computational operations required from it and the assumption is, therefore, valid.

1.3 Research questions

In this thesis the main focus is the performance of the access control solution:

Research question 1. *What kind of performance for a complete transaction can be achieved with our chosen embedded platforms?*

Research question 2. *How is the time taken by the transaction divided between different tasks?*

Research question 3. *What is the relative resource expense between different computational tasks, for example signature verification, hashing and parsing?*

There are also two potential performance improving techniques which I evaluate:

Research question 4. *What kind of performance benefits can be gained by using Certificate Chain Reduction?*

Research question 5. *Elliptic Curve Cryptography allows public keys to be compressed to almost half their original size: in what cases do compressed keys boost performance, and what are the advantages and disadvantages of using compressed keys in our system?*

1.4 Structure of the work

This work is organized as follows: Chapter 2 provides a brief background about access control, cryptography and the technology platforms used in this thesis. Then Chapter 3 presents the key architecture choices and Chapter 4 provides a detailed system description. Chapter 5 presents the measurement results obtained from the system, and said results are then analysed in more detail in Chapter 6. Discussion and future work are presented in Chapters 7 and 8. Finally, I present my conclusions in Chapter 9.

2 Background

In this section I present the five areas related to this thesis. The first topic is Access Control, a process of restricting the usage of resources. The second topic is public key cryptography, concentrating especially on the use of digital signatures and Elliptic Curve Digital Signature Algorithm (ECDSA). The third topic is Simple Public Key Infrastructure (SPKI) and how SPKI Certificates can be used for Access Control. The fourth topic is Embedded Systems, presenting a short overview on the broad performance range of different microcontrollers. The final topic is Bluetooth Low Energy, which is the technology chosen as the wireless communication platform of the system.

2.1 Access Control

Access control is a way to secure the *resources* we want to protect from unauthorized use. Authorized *users* are granted *rights* to use the resources. Typical example of a physical access control solution is having a lock on your front door, protecting your home, only allowing people with a proper key to enter. The act of giving the key is then equivalent to granting the right to access your home.

The process of access control can be split into 4 phases as shown on the figure 2 below: [20]

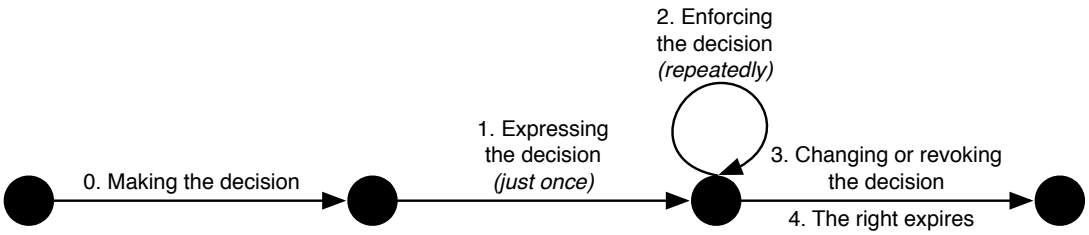


Figure 2: Phases of access control [20]

In *Phase 0*, the *issuer*, or the party controlling the resource or a right (e.g. parking service) decides to grant a subject (e.g. the driver) some permission to use the resource (e.g. the parking lot). In our case, an example would be a company deciding to grant an employee permission to use the company parking lot for the current year.

In *Phase 1*,the issuer creates the right, for example by creating and providing a certificate for the user, or in other words, expresses the decision. In our case, right would granted by the company and would take a form of a certificate proving that the user is allowed to access parking lot A.

In *Phase 2*, an entity protecting or guarding the resource verifies that the right granted to access it is still valid and that the entity attempting to use the right is the same entity the right was granted to. In our example, this phase would be performed by the Barrier. Phase 2, enforcing the decision, can happen multiple times.

Phase 3, revoking or changing granted rights, is optional. In many cases, being able to revoke permissions granted earlier is useful: For example, if the user was an employee and had permission to park at the company parking lot, such a permission would be revoked if the employee left the company.

Finally, if the right is limited in either its duration or usage, it will expire in *Phase 4* after the specified time period has elapsed or the uses have run out. In our case, this would happen in the following January.

There are three different ways to organize access control rights distribution digitally, of which examples are shown in Figure 3: the first one is to have an *access matrix*, where authorized users are rows and resources are columns. Intersecting cells will then contain the exact rights being granted. While Figure 3 only shows binary "access granted/denied" data, it would be possible to store very complex data in the matrix cell. In our example, it would contain the time period when the parking permit would be valid.

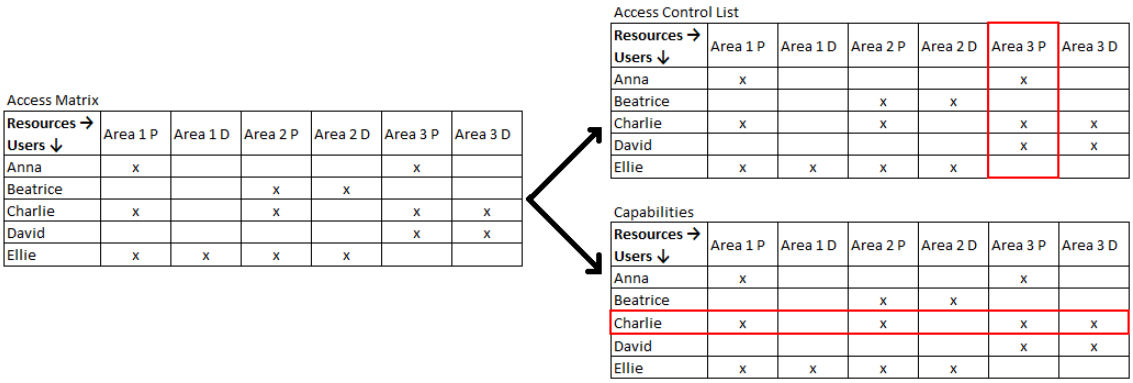


Figure 3: Access matrix, Access Control List and Capabilities

As the number of users and resources grows, it becomes evident that this approach is inefficient: the matrix grows very quickly to a tremendous size. Yet this matrix is usually sparse, with most of the cells empty, as typically the majority of users will only be granted access to a small part of the resources.

Two logical options to fix this problem emerged: *Access Control Lists*, where matrix column information is stored separately for each resource, giving us a list of users allowed to use a given resource and *Capabilities*, where matrix row information is stored on an user by user basis, giving each user a list of rights they have, also known as a *capability list*. In both cases, there is no need to store information about empty cells (shown as blanks in Figure 3) which can result in major reduction of required memory.

An Access Control Lists (ACL) is often stored in the trusted memory of the resource, as the resource will require access to the list to verify an user's right to use the resource. An advantage of ACL is that the list is always up to date and it is easy modify items in it, making revocation instant.

However, there are three main downsides to using an ACL setup: first, changes can only be made when the issuer has access to the list, so if the issuer or the list

is offline, no changes can be made to it. Second, if there are multiple resources that have to share same Access Control List, for example several doors of a building, they must all have a network connection to a server holding a master list that all changes are made to. While local copies of this master list can be made, there is still the problem of replicating changes between all the different copies of the list at each resource. Finally, last problem is that if the device responsible for verifying the right, the Barrier in our use case, has limited memory, the ACL list can grow so large that it cannot be stored in the device. A good example of this issue would be electronic locks used in large office buildings.

Capabilities, on the other hand, can be easily implemented as a system where each user provides the resource a proof that the user is authorized to use it. This has the advantage that new capabilities can be created and distributed without propagating this information to each resource separately, unlike ACL-based systems do. This gives Capabilities based approach a major advantage for distributed systems.

However, if the granted right is given to user to be presented to the resource for verification, it must be protected against possible tampering. There is an inherent disadvantage with the user presenting the right: while in an ACL setup, removing a granted right is very easy, a capability-based system requires some method for revoking granted rights. A typical way is to send a list of revoked capabilities to each verifier. Before granting access to the resource, the verifier checks if the presented capability is in the revocation list and denies access in that case.

2.2 Cryptography

Cryptography, and especially *digital signatures* enabled by asymmetric cryptography, also known as *public key cryptography*, is one of the cornerstones of the modern digital world.

Public key cryptography is based on two separate keys: a *private key* and a *public key*. The private key is kept secret by the entity creating the key pair, while the public key can be distributed freely. The private key can be used either to decrypt a message encrypted with the public key or to create a signature, whereas the public key can be used to verify a signature, encrypt a message to be decrypted with private key, or the public key can be used as an identity.

Hash functions are one-way functions which can be used to transform arbitrarily long data into fixed length output, where even a small change in the input data will cause large difference in the output data. A hash function is considered cryptographically secure when it is practically impossible to both calculate an inverse of the hash function and to generate a message with a given hash (collision).

So the combination of hashing functions and public key cryptography are the factors that truly make digital signatures possible:

- Only an entity possessing a private key can create a digital signature
- As each document has an unique hash, so will each signature be unique. This is not necessarily an one-to-one relationship: there are algorithms that provide multiple valid signatures for a single message.

- Anyone possessing the public key is able to verify the signature, and as long as we can identify the owner of the public key, we can identify the entity signing the message.

Digital signatures enable the creation of digital certificates (henceforth *certificates*), which are fixed form signed electronic documents, commonly used to prove ownership of public keys or rights. The majority of them work by binding two of either names, public keys or rights: this leads to three different types of certificates as shown on the Figure 4: identity certificates, authorization certificates and attribute certificates. These certificates contain at least the issuer, or the entity whose signature is used to guarantee the information within and the subject, or the entity about whom the certificate provides information.

In the scope of this work, digital signatures are used to fulfill two information security objectives: *Authenticity* and *Data Integrity* [26]. In our use case, Authenticity means that the Barrier can identify the original source of the certificate(s) sent by the Client and Data Integrity means that the Barrier can detect if the received data has been altered. Together, Authentication and Data Integrity allow the Barrier to verify that the Client requesting access to the parking lot is who he claims to be and that the certificates, and thus the permissions granted by them, have not been forged.

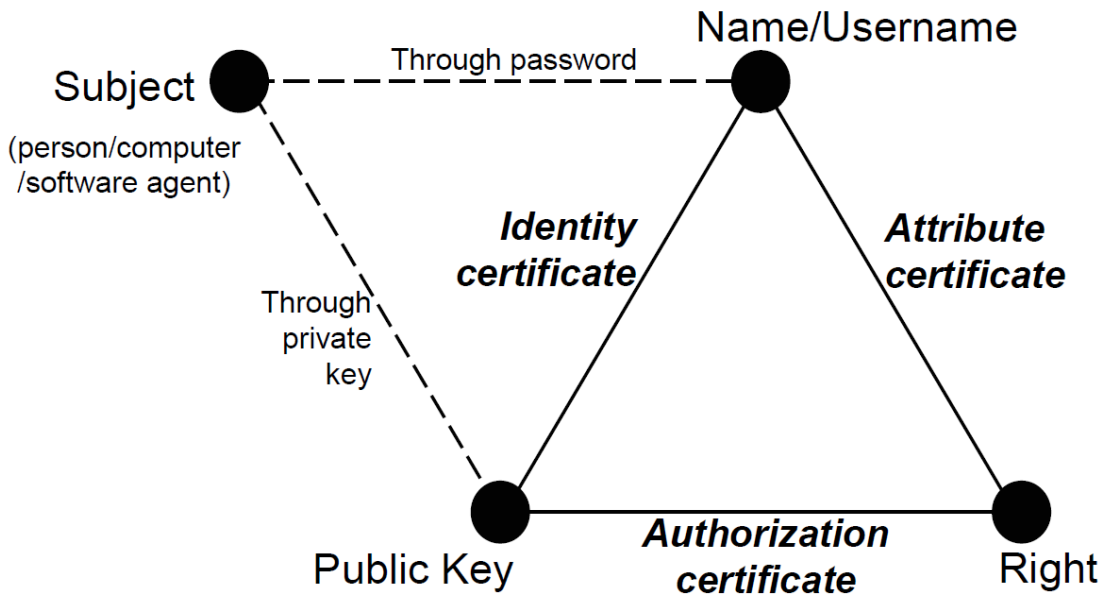


Figure 4: Three main types of certificates[20]

Identity certificates use the signature of trusted third party (issuer), usually a Certification Authority, to prove that the public key contained within belongs to a particular name (subject), which typically is a person's username or real name.

An authorization certificate binds a granted right to a subject's public key. Because the authorization certificate is bound to a public key and not a name, any

subject wishing to use the right must also provide proof that the public key belongs to the user, which is easily done by providing a service request signed by the user.

Third certificate type, attribute certificate, binds rights to a subject's name which is basically the same binding as Access Control List does. For the purposes of this thesis, I will concentrate on the authorization certificates and will not be discussing identity or attribute certificates any further.

2.2.1 Elliptic Curve Digital Signature Algorithm

Invented in 1985 independently by Koblitz [19] and Miller [27], Elliptic Curve Cryptography was a new way to implement public key cryptography. Where older asymmetric cryptography techniques like RSA are based on modulo mathematics [30], elliptic curve cryptography is based on elliptic curves over finite fields.

The main advantage ECC has over older cryptosystems is that ECC scales better as the required security level increases. As can be seen on Table 1, when symmetric equivalent encryption key length doubles, ECC key size also doubles, but in RSA required key size increases exponentially and the cost of cryptographic operations depend on the key sizes.

ECC also has the pleasing property that all cryptographic operations required for digital signatures (key generation, signing and verification) take roughly the same amount of time, unlike in RSA, where signature verification is much faster than signing, and key generation is very expensive. An example of this can be seen in the Table 2, where RSA key generation is 2 magnitudes more costly than signing, which is again 2 magnitudes more costly than verification, in comparison to ECDSA for which the most expensive operation, verification, costs less than 25% more time than cheapest operation, generating a key pair.

For the ARM Cortex-A8 processor, one can see from Table 2, that for equivalent security level, ECDSA takes approximately 6 times as long to verify a signature, but RSA on other hand takes 25 times as long to sign a message. For that processor, ECDSA is a better choice in systems where there is either a need to create keys relatively often, or key generation is time critical process, or in cases where there are less than 25 verifications per signature generation required. The performance profile of signing being far more expensive in RSA than verification is intended: relative costs of those two operations can be adjusted via choice of parameters.

Symmetric	RSA	ECC	Time to break in MIPS years
80	1024	160	10^{12}
112	2048	224	10^{24}
128	3072	256	10^{28}
192	7680	384	10^{47}
256	15360	521	10^{66}

Table 1: Key Comparison of Symmetric, RSA, ECC [15]

The elliptic curve public key consists of two numbers, x and y. One interesting phenomenon caused by the properties of elliptic curves is that for any valid public

Cryptosystem	Relative cost to generate a key pair	Relative cost to sign 59 bytes	Relative cost to verify 59 bytes
ECDSA, 256 bits	0.81	0.85	1.0
RSA, 3072 bits	4100	21	0.17

Table 2: Comparison of relative costs of equal security level RSA vs ECDSA on ARM Cortex-A8 @ 720 MHz[37]

key with given x, there are only two possible values for y, both of which can be derived from the value x. Effectively this means that the public key used in ECC can compressed to almost half of its original size, storing only the first number x and the knowledge which of the two possible numbers y is. For example, 256-bit ECC public key requires 64 bytes to store and transfer in uncompressed format, but only 33 (32+1) when using the compact representation as described in "Compact representation of an elliptic curve point" [17].

Having a smaller key size also helps in cases where there is a limited amount of memory available, as is the case in embedded systems, and transferring compressed keys requires less bandwidth. The effect of compressing public keys has on performance (Research Question 5 will be evaluated later in this work.

Elliptic curve domains are defined by a number of parameters. Certain parameter combinations are recommended by standardization organizations like National Institute of Standards and Technology (NIST) and Standards for Efficient Cryptography (SEC). SEC has defined two categories of curves over prime fields: r curves, based on verifiable random numbers and k curves which are based on Kobliz curves and are more efficiently computable.

In ECDSA, creating a digital signature is based on calculating the hash of the message to be signed and creating a pair of numbers based on the hash, the private key of the signee and a random integer k (where k needs to have certain properties). It is paramount that each signature has an unique k or the private key can be calculated from the signature [7]. This also means that if one signs the same message multiple times, each signature will be different.

2.3 Simple Public Key Infrastructure (SPKI)

SPKI is a Public Key Infrastructure that has been studied for 20 years, but whose standardization hasn't yet been finished[31]. While many aspects of SPKI have been researched, the studies relevant from our perspective concentrate on managing the use of authorization certificates [20], delegation [24, 10] and distributed systems [6, 21, 38]. A more detailed literate study is presented in the paper "Survey of certificate usage in distributed access control" [23].

There is some research that also evaluates performance [6, 32], the performance measurements presented in them are done on a PC [6] and might be extending SPKI with different technologies, for example Kerberos [32]. While Burnside et al. [6] discuss the implications of using authentication with embedded devices, their choice was to assume that embedded devices would not be able to perform public key

cryptography in a practical amount of time and to use proxy architecture instead, whereas the focus of this thesis is to evaluate if modern embedded systems are, in fact, fast enough to use SPKI.

SPKI focuses on authorization certificates, in which the resource grants certain rights to a subject, who in our use case is identified by a public key. The Subject then provides this certificate together with his or her signature to prove that he or she should have access to the resource.

For the purposes of this work, the contents of an SPKI Authorization certificate can be abstracted into a 5-tuple, whose elements are: [9]

1. Issuer: a public key identifying who is granting the permissions.
2. Subject: a public key identifying who is the entity to whom the permissions are being granted.
3. Delegation: tells if the Subject has the right to delegate the right further (in other words, issue new certificates)
4. Authorization: list of permissions given by this certificate to the Subject.
5. Validity: not-before and not-after dates, specifying the time interval when certificate is valid. Both dates are optional; a missing date implies no time limit in that direction.

This information must then be digitally signed by the issuer so its authenticity and integrity can be verified.

The SPKI specification draft also presents mechanisms for having certificates that are validated online. They could be used for example on usage based billing, where the user purchases permission to park for 100 hours at a parking lot. In our case, the permits are valid for defined periods of time and so further discussion of online validation mechanisms is considered to be beyond the scope of this work. Similarly, though Certificate Revocation is an important part of any certificate infrastructure, it is also considered beyond the scope of this work.

Delegation is a very important feature in SPKI Authorization certificates, as the user does not have to ask authorization directly from the resource or central entity. The right can be obtained from any entity that has been granted both the right and the ability to delegate it, resulting in a chain of certificates used to prove that the right has been granted to the end user.

An example of chain of trust can be seen in the figure 5, where a resource issues a Certificate for the resource owner, giving full permissions and delegation rights to the resource owner. The resource owner can then delegate a portion (or all) of these rights down to Service Provider via a certificate, Service Provider can then delegate these rights further downstream by granting a certificate to a reseller, who can grant a certificate to a Sales Point, which will be the entity issuing a certificate for our driver. Driver can present this certificate chain as a proof of the parking permission.

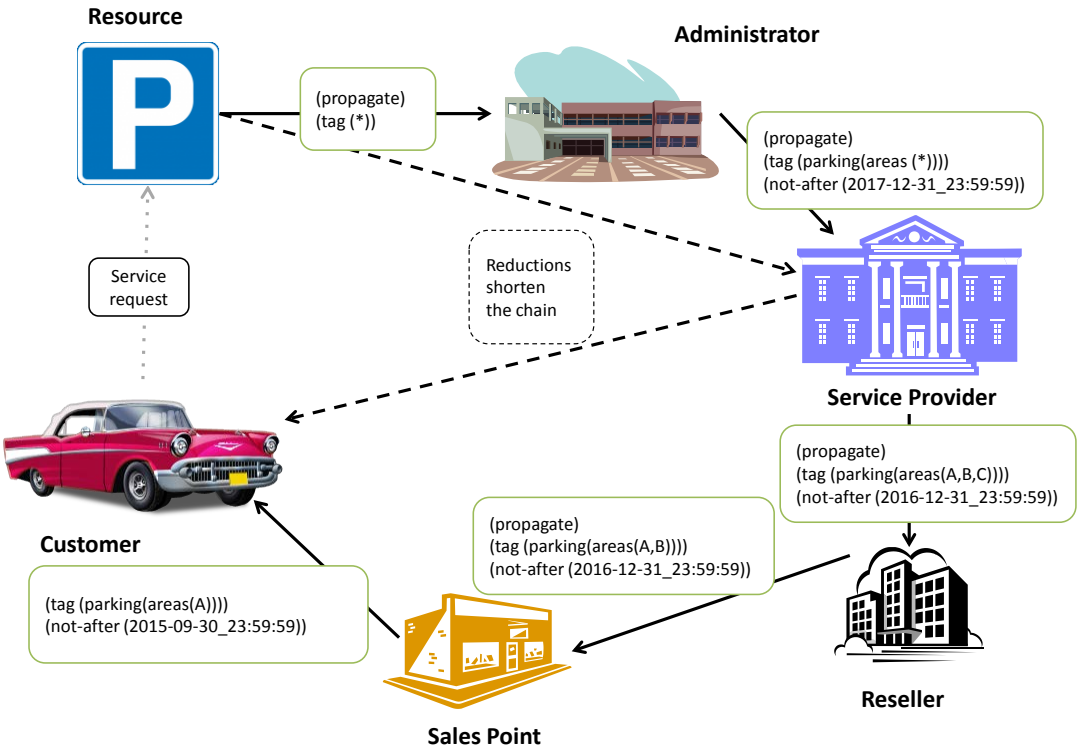


Figure 5: Chain of Trust
[22]

2.3.1 Certificate Chain Reduction

There are inherent disadvantages in using long certificate chains presented in the previous section: every time the right is to be used, the resource must validate each and every certificate in the chain separately again, even if the permission itself remains the same for multiple uses. For the user there is also a privacy issue: a long certificate chain reveals details about the user.

Of these two, the former is the far more important aspect in the scope of this work. For each certificate in the chain, a digital signature must be verified, and this process has a cost in computational resources. This poses a challenge for embedded systems that often constrained resources, as described later. For embedded systems, there is also a secondary consideration regarding the memory use of storing long certificate chains.

In Certificate Chain Reduction, the certificate subject presents the certificate chain and requests a new certificate from an upstream issuer. For example, instead of having a certificate chain Resource→Administrator→Service Provider→Reseller→ Sales Point→Customer, we can have a chain of Resource→Service Provider→ Customer, where certificates from Service Provider to Reseller, from Reseller to Sales point and from Sales Point to Driver are replaced by a new certificate issued by the Service Provider to the Customer.

Instead of having the Service Provider requesting Chain Reduction from the Resource, the Resource can perform an internal reduction when it receives a certificate chain that has the chain of Resource→Administrator→Service Provider present. The resulting Resource→Service Provider certificate is stored inside the trusted memory of the Resource and thus does not require signing, saving resources. As the Resource has to verify the certificate chain in any case, these internal reductions can thus be considered free [22].

Figure 6 depicts an example where a certificate chain of five certificates is reduced to two, by first having the Resource make an internal reduction of the chain to Service Provider and then having the Customer request a Chain Reduction from the Service Provider.

The system can also be constructed so that Customer does not have to present a full certificate chain to the resource: if the number of certificates issued directly by the Resource is small, it is possible to cache them in the trusted memory of the Resource so that the first certificate in the chain doesn't have to be presented. This is the underlying assumption in our use case. So, with one of my research questions concerning the performance advantages of using Certificate Chain Reduction (Research Question 4), this leads to two use cases: one where the Client provides four certificates to the Barrier, and another where only one certificate is sent.

Another option presented in the SPKI specification draft [31] is to have a mechanism, where the verifier would be presented with information on where to retrieve the missing certificates of the chain, but it is considered beyond the scope of this work.

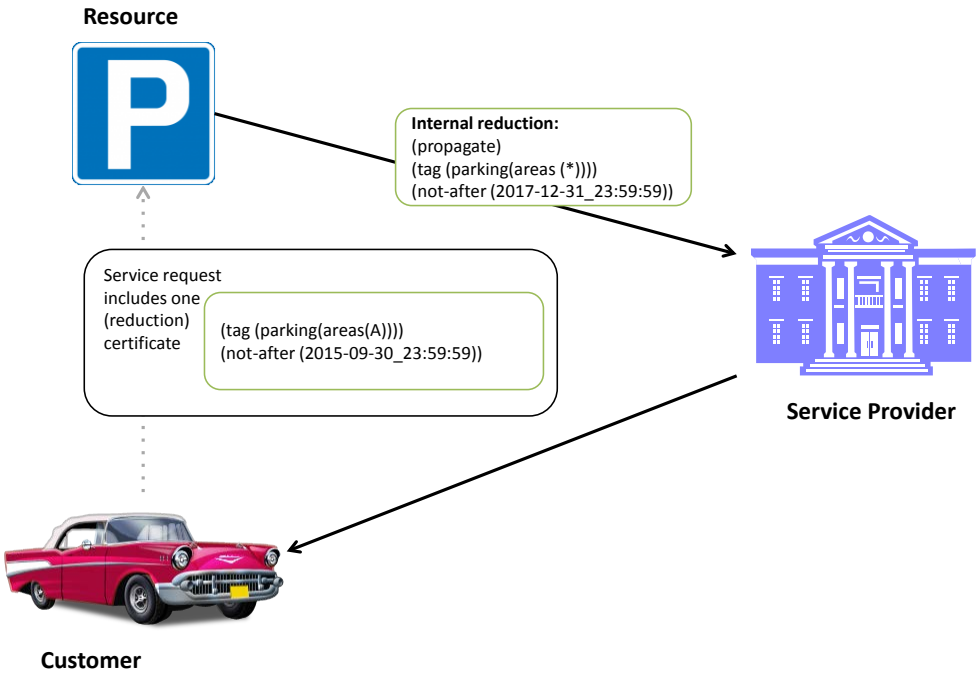


Figure 6: Chain reduction
[22]

2.4 Embedded Systems

One definition of embedded systems is that "Embedded systems are information processing systems embedded into enclosing products"[25]. Embedded systems are often implemented using microcontrollers as their main central processing unit, and microcontrollers have certain characteristics which make them significantly different from general purpose personal computers like desktops and laptops. They have three major advantages: microcontrollers have significantly smaller power consumption, often by multiple orders of magnitude. They are also significantly smaller in physical size as well as significantly cheaper. This combination makes embedded systems typical choices for distributed systems. There are of course trade-offs in performance which have been made to achieve these advantages. I will use STM32 family of ARM Cortex-M0 to Cortex-M7 to show the performance characteristics of typical microcontrollers: [33]

- Constrained CPU resources: microcontrollers are both clocked slower and do less work per processor cycle than desktop processors. Typical microcontroller would be able to perform between 30 to 400 DMIPS (Dhrystone Millions of Instructions Per Second, a unit of computational performance), whereas modern smartphone would perform between 7 000 to 30 000 and typical modern desktop processor would perform between 50 000 to 300 000 DMIPS.

- Constrained RAM: microcontroller memory is calculated in Kb, smartphone and PC memory in Gb. Typical memory amount in microcontrollers ranges from 4k to 320k. RAM and the memory controllers used in microcontrollers are also slower than their respective counterparts used in general purpose computers.

- Constrained storage space, usually Flash. Typical Flash sizes range from 16k to 2048k. Flash has limited write cycles which mean that it cannot be used for storing data that changes often.

When implementing software on microcontrollers, one has to take into account at least the following considerations:

- Microcontrollers are usually programmed via non-interpreted languages like C and C++, because interpreted languages require a relatively large amount of resources to process.

- There are significantly fewer ready-made libraries available for microcontrollers than for desktops. Although embedded systems outnumber desktops by a significant margin, the number of people developing software for embedded systems is significantly smaller.

- Even when using manufacturer-provided standard libraries, there usually is a lesser amount of abstraction layers available for programmers out of the box.

Usually embedded systems have certain auxiliary functions outside "normal" CPU computational functions added to them. In many cases they are the main reason why a microcontroller was chosen as an implementation platform.

Common examples of such functions include General Purpose Input/Output (GPIO) ports, hardware timers, communication interfaces ranging from Universal Asynchronous/Synchronous Serial Transports (USART) to Controller Area Network (CAN) interfaces and Real Time Clocks (RTC). Some even have true cryptographically secure Random Number Generators (RNG) and hardware accelerated cryptographic sub-processors. [34]

In this thesis I will study the performance of embedded systems (Research Question 1). Given how large the spread is in the performance characteristics in embedded systems, using both a low- and a high-end platform will help in evaluating the practical capabilities of the system. While the time taken for each different task will of course vary between platforms (Research Question 3), I believe that the relative expense of operations (Research Question 2) will most likely be similar between the two embedded platforms.

2.5 Bluetooth Low Energy

As will be explained in Section 3.1.2, I chose to use Bluetooth Low Energy (BLE) as the wireless communication link platform. Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a wireless technology operating on the same unlicensed Industrial, Scientific and Medical (ISM) 2.4 GHz band as classic Bluetooth and Wi-Fi. It was first introduced by Nokia under name of Wibree and later merged into Bluetooth Standard in 2010.

While Bluetooth Low Energy is not compatible with classic Bluetooth, Bluetooth Low Energy was designed so that the same antenna can be used both for Bluetooth and BLE transmissions. This design choice means that many modern chipsets supporting classic Bluetooth also support Bluetooth Low Energy. Reverse, on other hand, is not true, as classic Bluetooth requires significantly more resources to support it than BLE does. This means that currently Bluetooth devices fall into three categories: those supporting classic Bluetooth only, those supporting both classic Bluetooth and Bluetooth Low Energy and those supporting Bluetooth Low Energy only.

Bluetooth Low Energy is designed entirely from the perspective of low energy usage. In contrast to classic Bluetooth, which has 79 one MHz wide physical channels, of which 32 are advertisement channels, Bluetooth Low Energy only has 40 two MHz wide channels, of which 3 are reserved for advertising and the rest are used to transfer the main payload [13].

Because classic Bluetooth uses frequency hopping and contains a large amount of advertising channels, performing device discovery takes a relatively long time. In the worst case scenario, a classic Bluetooth device can take up to 10.24 seconds [13] to perform device discovery, though on average the value is around 2 seconds [39]. Bluetooth Low Energy is designed to perform device discovery significantly faster, of which results will be presented later in this work.

The three advertising channels used by BLE are designed to use parts of the ISM spectrum not used by the most commonly used Wi-Fi channels 1,6 and 11, as seen in the figure 7. The smaller number of advertising channels leads to device discovery taking less time, and thus using less energy. Advertising channels being situated in portion of the spectrum with less interference from Wi-Fi also makes it possible to use less transmitting power than would be required elsewhere in the spectrum.

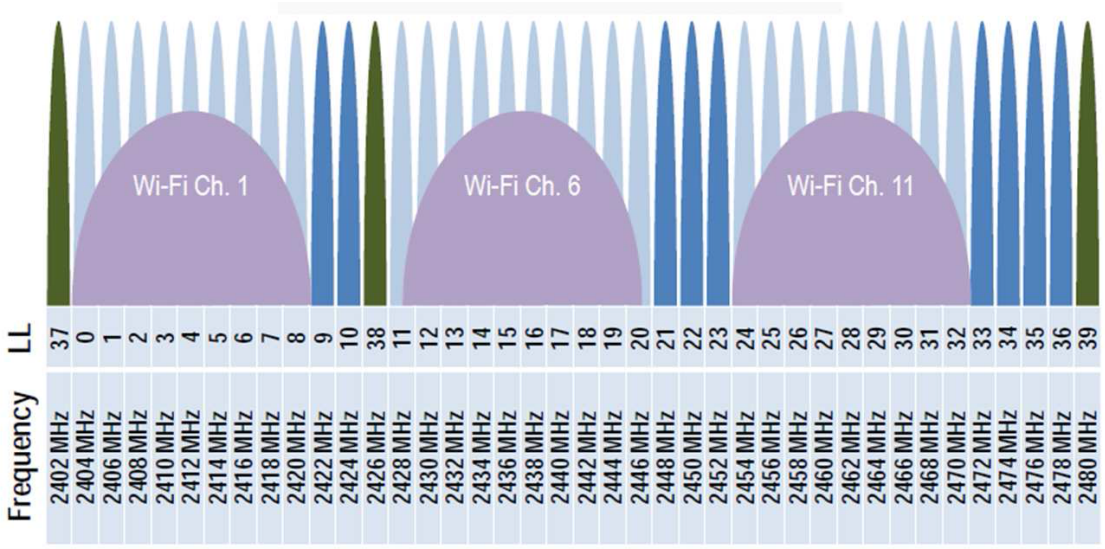


Figure 7: Bluetooth Low Energy channel mapping. Channels 37,38, and 39 are advertisement channels, the rest are data channels [16]

2.5.1 Profiles

Bluetooth Low Energy uses profiles for communicating between BLE devices. For the purposes of this work, profiles can be divided into two categories: Generic Access Profile (GAP), which is used to control advertising and connections and Generic Attribute Profile (GATT), which is used for transmitting information after the connection is established.

Bluetooth specification defines four specific roles for GAP: *Broadcaster*, *Observer*, *Peripheral* and *Central*, but for the purposes of this work only Peripheral and Central roles are interesting, as Broadcaster is optimized for only transmitting information and Observer only for receiving, but our application requires two-way transfer. Peripheral devices are optimized for having only a single connection at a time and are generally small, low power devices. Central devices, on the other hand, support multiple connections and a Central device is always the one initiating connection with a Peripheral device. One device can support multiple roles, but for the purposes of this thesis the Barrier is a Peripheral device and the Client a Central one.

Advertisements are used to broadcast small amounts of information by a Peripheral device to all listening devices, but they are also used to discover or connect to the Peripheral device. So in our case, the Barrier is the device sending Advertisements

Peripheral devices use GAP to broadcast advertisement messages on the 3 advertising channels. Advertising works by having the Peripheral set an advertisement interval, ranging from 20 ms to 6 s, and having the Peripheral transmit an advertisement packet once per interval. Usable maximum payload for advertisement is 31 bytes: the Central device can request more data via Scan Response request, which allows another 31 bytes of payload to be broadcast without opening connection, as shown in the figure 8.

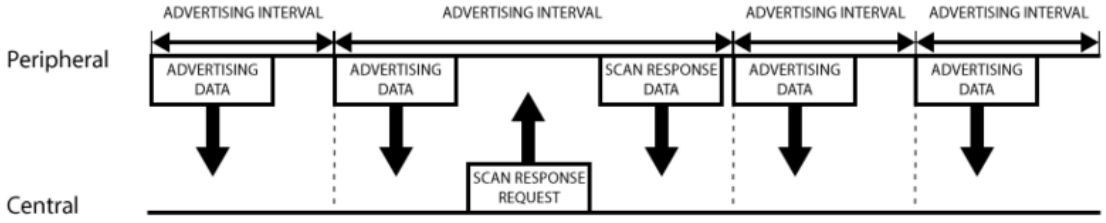


Figure 8: Bluetooth Low Energy advertising message [36]

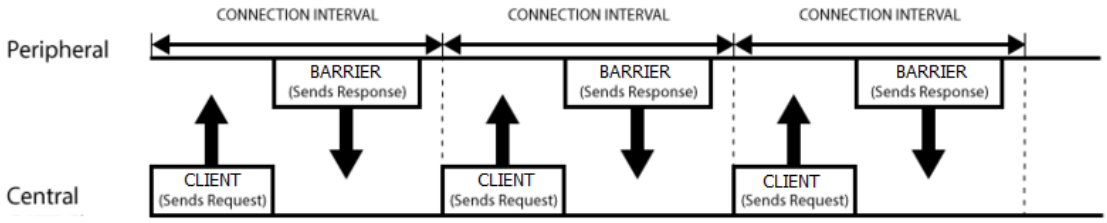


Figure 9: Example of communication between Central and Peripheral [36]

To establish a two-way communication, a Central device replies to an Advertisement by sending a connection request to the Peripheral device, after which a connection is negotiated and established. One of the things that has to be agreed then is the Connection Interval, which defines how often the Central device (Client) tries to connect to the Peripheral one (Barrier). During this time, Connection Interval parameter is specified, though it can be updated at any point later.

After the connection is established, payload data can be sent using GATT, with a high level example shown on figure 9. Figure 9 does not show payload at message level of detail: there can be multiple messages transmitted during single Connection Interval.

The Bluetooth Low Energy claims to have maximum throughput of 236.7 kbps [12]), achievable throughput is much lower. Bluetooth 4.1 specification states that the minimum Connection Interval is 7.5 ms, maximum of six messages being sent per Connection Interval and maximum payload per message being 20 bytes [13], this leads to maximum theoretical throughput of 128 kbps, though in practice the amount of messages transmitted in each interval is often dependent on the interval length and both the minimum interval length and the maximum number of messages transmitted per interval are device specific.

Recently released Bluetooth 4.2 Specification defines a Data Length Extension, which increases maximum message payload length by a factor of 10 to 200 bytes per message, though actual throughput is expected to increase only by factor of 2.5 as less messages can be transmitted per interval when using it [14].

3 Architecture choices

In this section I discuss the key architecture choices for the implementation. I have grouped these choices into two categories: platform and architecture. As the choice of the platform will restrict choices made for the software architecture, I will discuss the platform aspects first.

3.1 Platform

These include my choice of embedded platform used for the Barrier, smartphone to be the Client and Bluetooth Low Energy as the wireless technology used for communication.

3.1.1 Embedded platform

I chose to use ST Microelectronic as the embedded device supplier, as their products were readily available from multiple different product categories. To better understand how the performance scales (Research Question 1) I decided to implement the system on two microcontrollers: one low-end, one high-end. For low-end I chose to use STM32F051R8T6 ARM® Core M0 processor running at 48 MHz with 64 kb of Flash and 8 kb of RAM (38 DMIPS), and for high-end STM32F407VG, 168 MHz ARM® Core M4 processor with 1024 kb of Flash and 192 kb of RAM (210 DMIPS)

Both platforms also have certain integrated peripherals, like Real Time Clocks (RTC) allowing them to keep track of time, multiple Universal Asynchronous/ Synchronous Transmitters (USART) for sending and receiving serial information, Cyclic Redundancy Check calculators etc.

F4 has a random number generator (RNG) implemented in hardware, allowing it to be used for secure generation of public/private key pairs and for signing. F0 does not have one, so for cases where random number generation is required I will be using pseudo-RNG instead. In this implementation, random number generation is required from the Barrier for sending signed receipts to the Client.

While F4 has practically limitless memory (both Flash and RAM) as far as this application is concerned, F0 is very constrained in this regard, an issue which had to be taken into account in the implementation.

Standard libraries provided by ST will be used to provide low-level abstractions for programming the embedded systems, but while they are abstractions, they are not high-level abstractions: for example, standard library provided by ST only allows sending one character at a time via USART; to send complete strings, another abstraction layer has to be implemented.

Dynamic memory allocation is not used in any software implementation in the Barrier for three reasons: 1) it can lead to non-deterministic run times 2) it increases amount of bookkeeping that has to be done 3) dynamic memory allocation opens up the possibility of memory leaks, which are especially hard to debug in embedded systems. Barrier also pipelines computational operations so that they can be done simultaneously while receiving information sent by the Client.

3.1.2 Wireless communications technology

The candidate wireless technologies for connecting the Client and the Barrier were Bluetooth Low Energy (BLE), classic Bluetooth, ANT, Zigbee, Near Field Communication (NFC) and Wi-Fi. I chose Bluetooth Low Energy, as it filled all of our requirements, though it does have a slightly low throughput.

Classic Bluetooth would otherwise have been a good candidate, especially considering the significantly higher throughput of 2.1 mbps it possesses. However, classic Bluetooth has an average pairing time of over 2 seconds [39], which would have made it impossible to reach our business goal of completing the transaction in less than 1.5 seconds (Business Requirement 6).

Other technologies were discounted for several different reasons: Neither ANT nor Zigbee are integrated in modern smartphones (Business Requirement 3). While there are phones with integrated NFC, they are relatively rare and NFC range is only few centimeters (Business Requirement 5). Finally, setting up a Wi-Fi access point requires more expensive hardware than Bluetooth Low Energy does, obtaining IP-addresses via DHCP and other relevant procedures to enable communications can take many seconds and a Wi-Fi setup would also have required more security infrastructure as Wi-Fi has significantly higher range than BLE does: while a range of up to 10 meters is useful for our case, exceeding that range might become a liability.

For the Bluetooth Low energy communications chip I chose to use nRF8001, as it was readily available with manufacturer provided libraries. As the GATT profile I chose to use a custom UART data transfer profile that was provided by Nordic Semiconductor. Packet level message acknowledgments were not used as they would reduce throughput to one third. Nordic Semiconductor had also provided example code for using the custom UART profile for serial data transfer on Arduino microcontroller platform.

3.1.3 Modular architecture

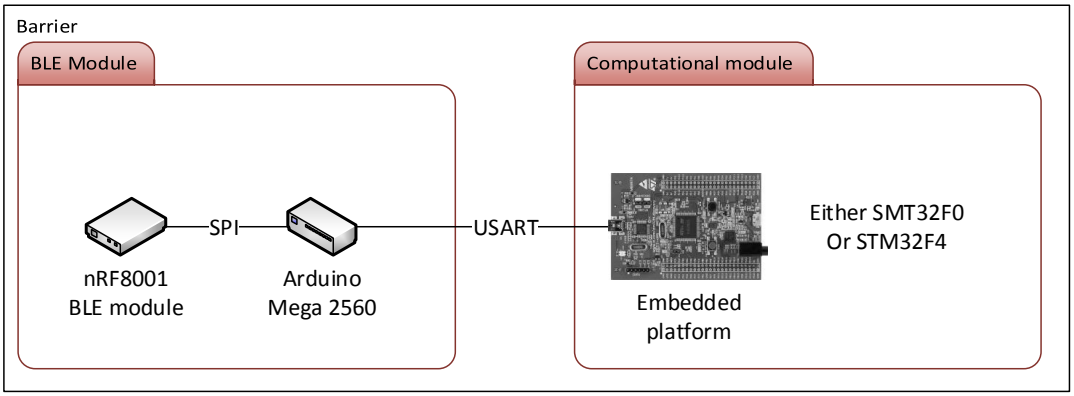


Figure 10: Barrier architecture

To better support using two separate microcontrollers, a modular architecture was used for the Barrier as shown in the Figure 10. Instead of integrating the Bluetooth Low Energy (BLE) chip nFR8001 directly into the embedded device, I chose to make a separate BLE module in which the nRF8001 is connected to Arduino Mega 2560 via Serial Peripheral Interface (SPI) bus. The BLE module then communicates with the embedded device in the Computational module via Universal Synchronous/Asynchronous Serial Transport (USART). This had the added benefit of not requiring us to port neither the libraries nor the UART-data transfer code to our embedded devices.

This modular architecture allows us to make changes in either portion of the Barrier without requiring further changes to the other, isolating the embedded platforms from each other, as long as the interface between the modules stays the same. This allows for easily changing the embedded device from F4 to F0, or it could be used to change the communications module to use completely different wireless technology without requiring changes to the embedded system.

3.1.4 Smartphone

Client software running on a smartphone was a business requirement (Business requirement 4). I chose to use LG Nexus 5 running on Android 4.4.4 as it was the current reference Android design at time the of implementation.

Per our scope (Section 1.2), I concentrate on evaluating the performance aspects of the Barrier and therefore certain features are not implemented in the Android application. For example, while the application does receive the nonce from the barrier, it does not generate or sign the service requests on the fly: instead the Client sends pre-generated service requests with embedded certificates, both of which have been made in advance, with separate PC software and scripts written for this task.

As stated earlier, I believe this will not affect the performance of the system in any measurable way as modern smartphones are at least a magnitude faster (7000+ DMIPS) than even the fastest embedded platform I used (210 DMPS), so based on the measurements, it is safe to assume that signing the service request takes less than 10 ms. Considering that the smartphone has at least 40 milliseconds (2x advertising interval) of time to do the service request generation and signing, not implementing these on the Client does not affect the performance measurements of the Barrier.

Later in the Results section (Section 5.4) I show that the Client has even more time, on average 230 milliseconds, to perform these operations.

3.2 Software and Architecture

Under this section I will discuss architectural choices regarding software implementation: Elliptic Curve Digital Signature Algorithm and encoding used for SPKI Certificates and service requests.

3.2.1 Elliptic Curve Digital Signature Algorithm (ECDSA)

Authorization certificates and the service request have to be digitally signed. I chose to use Elliptic Curve Digital Signature Algorithm instead of RSA, because in typical cases the resource cost of signing using ECDSA is at least a magnitude cheaper than signing using RSA [37].

Two main choices regarding ECDSA had to be made at this point: What key size to use and what parameters we wanted to use for the curve. There is a trade-off between increased key size and performance, and using a too large key size would be detrimental from the performance point of view. NIST recommends using a minimum key size of 224 bits from year 2014 to 2030 and 256 bits from year 2031 onwards [2]. To test the performance limits of the embedded system, I chose to use 256-bit keys.

Next choice was on what curve to use: Standards for Efficient Cryptography Group support using both so-called Random and Koblitz curves [8] and I chose to use secp256k1 (Koblitz) curve, as operations on it are faster to calculate than operations on secp256r1 (random) curve [4]. The downside of using Koblitz curves is that they are slightly easier to brute force. However, the effect is small, reducing the time required by factor of $\sqrt{3}$ [4] and considering the key size we are using this should be a non-issue.

Implementing Elliptic Curve Cryptography efficiently would have been far beyond the scope of this work, and thus I decided to use an open source library called uECC [18] for all ECC functions. uECC has been designed to work with microcontrollers and even has assembler optimizations that would work with F0.

uECC requires using 256-bit hashes for signing when using 256-bit key length, which limited our hashing algorithm choice. This, combined with the need for security, led me to choose SHA-256 as the hashing algorithm, as it is an industry standard [2] and its predecessor, SHA-1, has already been deprecated because of security issues [28].

In Elliptic Curve Cryptography, public keys can be stored and transmitted either in uncompressed or compressed format. For 256-bit public keys, using compressed format will save 31 bytes of space for each public key stored or sent [17]. My implementation uses compressed keys to evaluate their effect on performance.

3.2.2 SPKI Certificate and service request encoding and specifications

For all transmissions there needs to be an encoding for the data: a standard on how the information is represented, agreed by all of the parties participating in the transmission. For SPKI, two different encodings are already defined in literature: S-expressions [31] and XML-encoding [29].

Both of these are text encodings: in other words, they transfer information only via "printable" characters, making them human-readable and thus easier to debug. This has the downside of taking a relatively large amount of space, because text encoding uses only 7 bits of the 8-bit byte. Space usage is increased even more by verbose field descriptions.

I chose to use Canonical S-expressions, which are a (proper) subset of S-expressions: the main difference is that Canonical S-expressions are easier to parse than "classic" S-expressions as they do not allow white spaces between fields.

XML-encoding was not chosen as it would have used even more space and XML-parsers are relatively large and resource intensive: for example, TinyXML2, an open source XML-parser designed for embedded and other low resource environments, is around 4500 Lines of Code long, and is not actually even fully featured [35].

To keep the information wholly human readable, all numbers were transmitted in hexadecimal encoding, which also wastes space: hexadecimal encoding requires sending 2 bytes to send 1 byte of information.

We could have saved space by using base64, which on average requires sending 4 bytes for each 3 bytes of information sent, but base64 is not as easy to parse for humans. Also, it is slightly harder to convert numbers to base64 or back, than to convert numbers to hexadecimal.

If transmission times are found to be a performance bottleneck, it is possible to create new specifications and change to using a more dense encoding in the future, though it might require a significant amount of work.

Of course specifying the certificate encoding is not enough, as it specifies only how the information is represented, not what information is required. To complete a transaction, a Client must provide a signed service request with an embedded certificate chain to the Barrier. In addition to those, a Barrier must provide a signed receipt for the Client when the transaction has been completed. This leads to having to specify four message portions: service request, certificate, signature and receipt. Their specifications for our use case are found in the Appendix A, but in essence, the message portions store the following information:

A service request contains the following fields: which parking lot does the user want to park at (Parking Area ID), who the user is (public key), a cryptographic nonce to prevent replay attacks, how long the service request is valid, and the length of the certificate chain. While certificates are technically embedded inside the service request, they are discussed separately.

A certificate contains the following fields; who has issued the certificate (public key), who is the subject (public key), does the subject of the certificate have the right to issue new certificates (delegation), what parking permissions does this certificate grant, and for what time period (from where to where) this certificate is valid.

A signature contains the following fields: a hash of the message portion (service request or certificate) the signature refers to, who is signing the message portion (public key), and the hash signed cryptographically by the signee.

A receipt contains the following fields: a hash of the received service request and a reason code specifying whether the service request was accepted or denied.

4 Implementation description

In this section I present a more detailed explanation of the system components, the Client and the Barrier. Following that, I give a more detailed overview of the transaction, which starts with the car arriving to the entrance of the parking lot, and ends with the Barrier granting access to the parking lot. Finally, the internal workings of the software are discussed, ending with the considerations caused by the very constrained resources of F0.

4.1 System Components

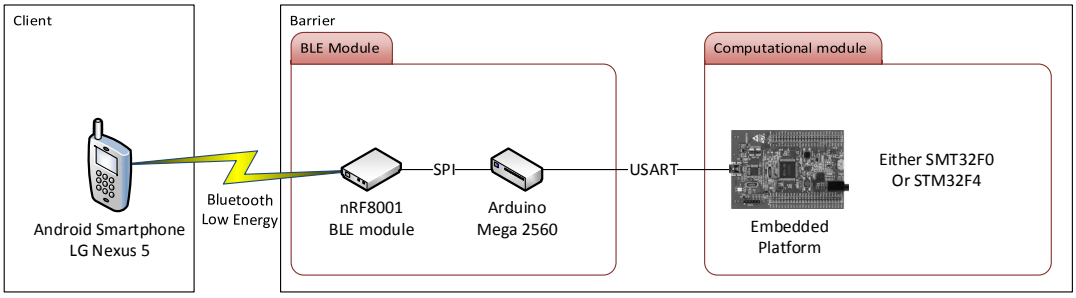


Figure 11: System architecture

As seen in the Figure 11, the system is split into two main components: the Client and the Barrier, with the Barrier being further divided into two modules, Bluetooth Low Energy module and Computational module. The main responsibility of BLE module is to send Advertisements (containing nonce and Barrier UUID) and to work as a communications bridge between the Client and Computational module, which is responsible for all non-BLE related tasks. A more detailed overview of the component responsibilities is given on Table 3.

To better evaluate the performance of the Barrier, it was implemented with two different computational modules: one built with STM32F0 Discovery Board (henceforth "F0") and one with STM32F4 Discovery Board (henceforth "F4").

BLE module consists of Adafruit nRF8001 BLE chip connected to Arduino Mega 2560 via SPI bus that is clocked at default rate 4 MHz, allowing transmission speeds of up to 4 mbps. This interconnection has a far higher speed than the theoretical 1 mbps symbol rate of Bluetooth Low Energy has, meaning it will not slow down any transfers. Communication with the Client is handled by publishing an UART-type interface that allows the Client to send and receive messages, each containing 20-byte payload.

There is an another communication bridge inside the Barrier, between BLE module and Computational module. This one is implemented via standard USART. This USART bridge does not cause a bottleneck in communications either, as the

Task	Component	
	Client	Barrier
Send advertisements containing nonce and UUID		X
Present user an interface to use the system	X	
Send signed service request with embedded certificate chain	X	
Received signed service request		X
Check that service request is correct and contains a valid nonce		X
Verify integrity and authenticity of the service request/certificate chain		X
Evaluate the certificate chain		X
Allow/Disallow access to the resource		X
Send signed receipt to the Client		X
Receive receipt	X	

Table 3: Tasks and the components responsible for them

maximum speed of the USART link between the modules is 1 mbps, significantly higher than the maximum achievable throughput of the BLE link.

4.2 Transaction overview

As stated in Introduction (Chapter 1), our use case starts with the driver arriving to the entrance of the parking lot. For our purposes, the transaction begins when the Client initiates device discovery and ends when the Barrier has made a decision about granting access to the parking lot and signals it to the user by opening the gate, and/or via traffic lights. From the driver's point of view, the transaction is now complete: even though the Barrier still has to generate, sign and send the receipt that to the Client, this will be done while the gate opens, before the driver has had time to enter the parking lot.

I have divided that transaction into following tasks, as shown on the Figure 12:

0. The Barrier sends advertisements with 20 ms interval, repeating until connection is established.

\# Transaction begins

1. The Client initiates device discovery and finds the Barrier.
2. The Client connects to the Barrier.
3. The Barrier sends an ACK to the Client, signaling that the connection parameters have been negotiated.
4. The Client sends a service request with an embedded certificate chain to the Barrier.
5. The Barrier starts processing message portions as they arrive, instead of waiting for the Client to finish transmitting the whole message.
   (a) A message portion is parsed, its type recognized and data fields extracted.

   (b) If the message portion is a signature, it is used to verify the message portion it refers to.

6. Having received and verified the final signature (for the service request), the Barrier evaluates the certificate chain and opens the gate if everything was correct.

Transaction ends

7. The Barrier generates, signs and sends the Client a signed receipt.

If at any point there is missing information, or an invalid signature, the Barrier sends an error message to the Client and aborts the process. Even in this case, it sends a signed receipt to the Client, after which the Barrier returns to sending advertisements.

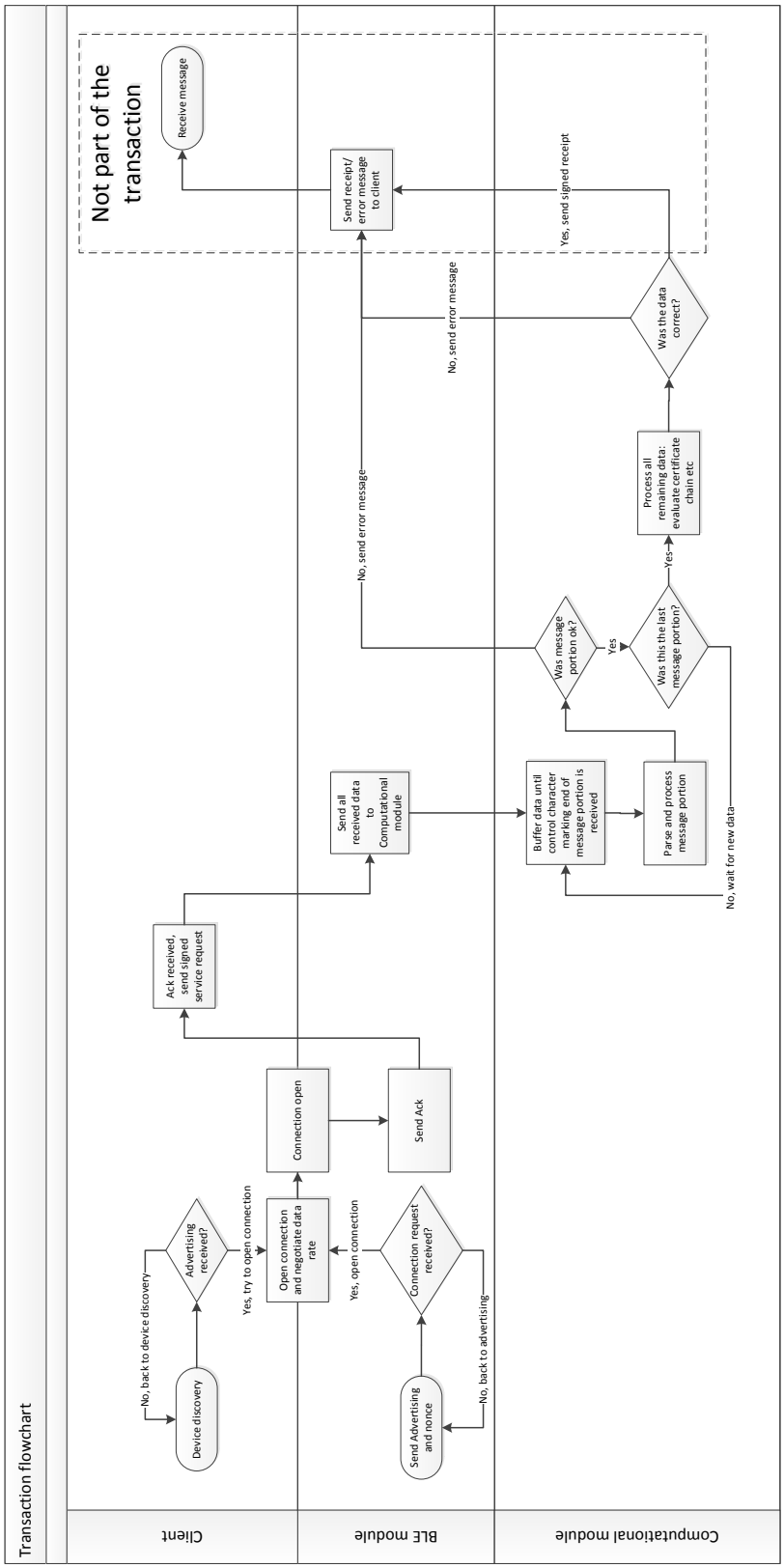


Figure 12: Transaction Flowchart

4.2.1 Messages

There are six different types of messages used in the system, with different functions.

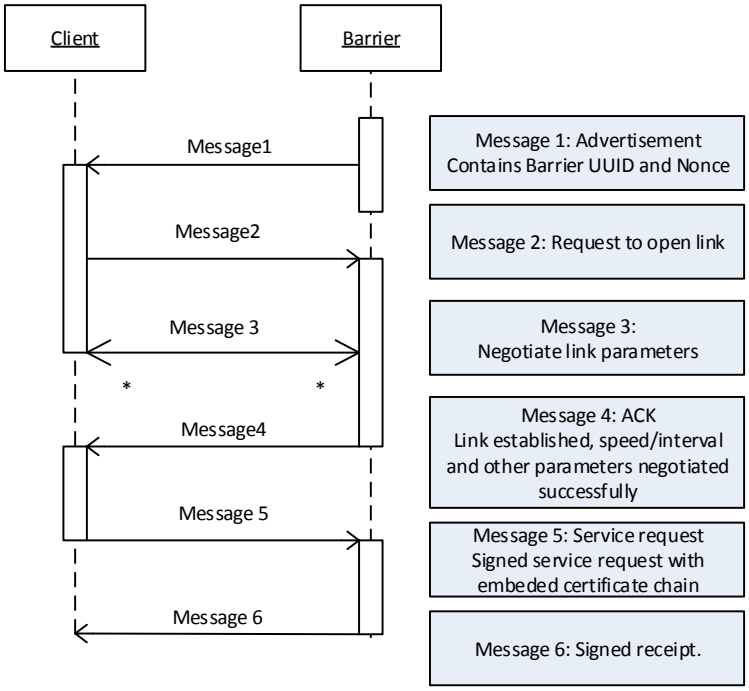


Figure 13: Messages exchanged in the system

Figure 13 shows the messages that are exchanged by the Client and the Barrier. Content of these messages is detailed below, with messages 1 - 5 being part of the transaction and message 6 being sent as the gate opens.

Message 1 Advertisement sent periodically (every 20 ms) by the Barrier. Payload of the advertisement message contains the nonce and UUID of the Barrier, from which the Client recognizes what parking area the Barrier belongs to.

Message 2 Request to open link (initiate connection)

Message 3 Actually consists of multiple messages where Client and Barrier negotiate all link parameters, Connection Interval of 7.5 ms being the most important one from the perspective of this work.

Message 4 The Barrier sends "Acknowledged" message (ACK) when the Connection Interval has been negotiated to correct level.

Message 5 consists of actual payload the Client wants to send to the Barrier: signed service request with an embedded certificate chain, proving that the Client both wants, and has right to, access services guarded by the Barrier.

Message 6 Is a receipt signed by the Barrier, consisting of a hash of the service request, a reason code for granting or denying access and a signature.

Message 5 is encoded as Canonical S-expression and consists of a signed service request with an embedded certificate chain. The certificate chain consists of signed certificate(s). **Message 5** is split into following types of message portions: service request, certificate(s) and signatures. Both for the service request portion, and for each certificate portion, there is a corresponding signature portion.

So the structure of **Message 5** is following:

```
Service request
    Certificate 1
    Signature 1
    ...
    Certificate N
    Signature N
Signature N+1
```

Altogether there are 2N+2 message portions in the **Message 5**, where N is the length of the certificate chain. In our normal use case, while the system uses 5 certificates as per Figure 5, the Client only has to present 4 certificates to the Barrier or 1 certificate, if Certificate Chain Reduction is used.

4.3 Software overview

Creating the software for the microcontroller required writing only slightly over 3000 Lines of Code, as all low-level functions were implemented using standard libraries provided by ST and only timing, USART and RNG required significant extensions to be written for them. For both SHA-256 and ECC, there were freely available libraries, either GPL or BSD-licenced. For comparison, these two libraries were around 200 and 2500 LOC respectively. The author of uECC also provided a C-program for testing the library, and a modified version of that was used for running the cryptography performance tests.

For Bluetooth LE communications, programs or libraries providing basic functionality for USART type data transfers were also available and used: Nordic RF, the manufacturer of the BLE chip used in our BLE module had ready made libraries for AVR/Arduino and example code that was modified to work for our requirements. Similarly, Android software that had basic UART-BLE transfer functionality was available on GPL license, and it was used as basis of the Client software in the system.

I have split the main functions of the Barrier software into following tasks:

1. Communication: sending and receiving communications via USART
2. Buffering: storing received messages into buffer for later processing
3. Parsing: extracting relevant information from the messages
4. Cryptography: calculating hashes from incoming messages, decompressing public keys and verifying signatures.
5. Evaluation: Evaluating certificate chains

4.3.1 Communication, buffering and parsing

Communication was implemented using interrupts for two reasons: first, it would save clock cycles as we wouldn't need to poll USART interface periodically even when we were not receiving communications and second, it would allow us to implement better power saving schemes if those would be required in future.

For buffering, a classical circular buffer was used. For our faster platform, F4, the buffer was 10 message portions long, and thus easily able to store our larger use case, where certificate chain was of length 4, completely in the buffer. Unfortunately our slower platform F0 was resource constrained, and the buffer had to be reduced to storing only 4 message portions in it, which meant introducing delays in the Bluetooth Low Energy module as described later in Section 4.3.4.

To ease parsing, control characters were inserted in the data stream in the Client code, marking the end of each message portion. After a complete message portion was received in the data stream, that portion was then stored into the circular buffer for later processing. Each message portion was then parsed as a simple C-string, using standard C library functions to split the message into tokens, from which the required data was extracted. As per best practices, functions used for data extraction are protected against buffer overruns, and thus malformed fields will not cause writes to wrong memory areas: instead, malformed data field will result in message verification failing in a later step.

4.3.2 Verification

First step when a certificate message portion is received, is that a SHA-256 hash is calculated from it and stored. Because the embedded certificate chain is considered to be part of the service request, a separate hash of all received message portions (excluding final signature) is also calculated and is used to verify the service request.

So the verification process has 4 steps:

1. Calculate the hash for each received message
2. if the message portion was a service request, evaluate fields in it to save time in cases where there is a problem with the service request.
3. Compare the calculated hash against the hash received in the signature message portion.

4. Decompress the public key and verify the signature.

While a hash can be calculated immediately after the message portion has arrived, comparing it to received hash, decompressing the public key and verifying the signature can only be done once the signature message portion has been received.

The fields of the service request can be evaluated immediately after it has been parsed, as long as its signature is verified at some point. As evaluation is computationally cheap compared to verifying a signature, doing evaluation immediately after receiving the service request portion allows the Barrier to an send error message to the Client if a problem was detected, without the performance penalty associated with either verifying the signature or waiting for the signature to arrive.

Service request evaluation consists of the following: service request subject is the Barrier (i.e., the Barrier was the intended recipient of the service request) and service request is intended for the parking area that the Barrier is part of (Parking Area ID matches with the Barrier's). After that the Nonce of the service request is compared to the one the Barrier expects to receive (Nonce is present to prevent replay attacks). Finally, the request validity period is compared against Barrier Real-Time Clock.

The third step, done after receiving the signature message portion, is used to save time if the message has been corrupted in transmit. In it, a calculated message portion/complete service request hash is compared against the hash stored in the signature. This is not a security measure: any corruption in the message would be detected when verifying the signature in any case, but comparing hashes is at least 5-6 magnitudes faster than verifying a signature is.

In the fourth step, the public key in the signature message portion is decompressed and the signed hash stored in the signature message portion is verified via the use of the decompressed public key against the calculated hash of the message.

As stated in an earlier section (Section 3.2.1), an open source library was used for decompressing public keys, verifying signatures and signing messages. Same library was also used in the PC application to generate keys and sign messages. uECC [18] was written so that key generation and signing functions would be resistant to so called "timing attacks" [5], where time required to perform a cryptographic function leaks information about the key used in it. uECC also contains optimizations written in assembler for Arm Cortex-M0, whose performance effectiveness I will evaluate later in this work.

4.3.3 Evaluation

Finally, after all message portions have been verified, certificate chain is evaluated in the following steps

1. Certificate chain integrity is checked: subject of certificate N must be issuer of Certificate N-1. The issuer of the first certificate must be a subject of a certificate stored in the trusted memory of the Barrier.

2. Intersection of the validity periods is calculated and the result is verified against Barrier RTC.

3. Delegation chain completeness is verified (all, but final certificate in the chain, must have delegate bit set to 1)

4. Intersection of the rights for permits specified in the service request is calculated. If the resultant is not empty set, permissions intersection is then verified that the parking lot guarded by the Barrier is included in it and that the intersection allows parking at the current time, checked against the Barrier RTC.

4.3.4 F0 considerations

As F0 had significantly less processing power and memory than F4, certain adjustments to both Bluetooth transmission and Computational module code had to be made:

1. Because of constrained RAM, F0 can only process and store certificate chains of length 5 or less. The current limit in F4 is 10, but could be increased trivially to 50 or more.

2. Because of constrained RAM, F0 has a buffer size of only 4 message portions.

3. Standard library USART interrupt implementation in F0 was causing exceptions if it received transmissions while doing buffer operations (copying a message portion to or from the buffer). To fix this issue, the BLE module was modified to add a small 10 ms delay after each message portion was transmitted. This delay was not optimized to be as small as possible.

4. Because of the small buffer size and slow processor, using standard transmission rates would have caused the F0 message buffer to overflow. To avoid the overrun, the Bluetooth USART bridge was modified to change the delay added in item 3 to 100 ms after 4th message portion was received.

These changes were implemented in a fashion that allows turning them on and off easily via configuration parameters.

No changes were required for the Smartphone client to adjust for using F0 in the computational module.

5 Results

In this section, I present the performance results of the system. The measurements are split into four different groups:

- Message parsing and certificate evaluation
- Cryptography, i.e. hashing, decompressing public keys and verifying signatures
- Bluetooth Low Energy transmission times, including device discovery and time to open connection
- Transaction performance

These in turn can be split into measurements done on F0 and F4. All measurements were done for both F0 and F4, except for time required to perform Bluetooth service discovery and opening Bluetooth connection, as neither of those are dependent on the Computational module.

5.1 Methodology

I used three different methods to perform the measurements, using the Client, the Bluetooth Low Energy module and the Computational module.

The Client was used to measure the following values: Service discovery, opening connection and total time taken for transaction. The measurements were done using internal elapsed milliseconds counter. "Device discovery" is the time taken by the Client from starting to search for Advertisements, to it first receiving Advertisement from the Barrier. "Opening connection" is the time between the end of device discovery and the Client receiving first "Connection OK" message from the Barrier.

The Bluetooth Low Energy module was used to measure the time spent transferring information, and the measurement was done via the internal interrupt-based millisecond counter. Time spent transferring information was defined as the time between receiving first payload from the Client and receiving the last one.

The Computational module was used to measure all computational operations, the generation of signed receipt and the time spent from receiving first payload from the Client via BLE module to the Barrier having completed the evaluation of the certificate chain. Transaction length can then be calculated by adding the time between receiving first payload to completing certificate chain evaluation, to time taken by "device discovery" and "opening connection".

Timing measurements on the Computational module were measured using the "Systick" interrupt counter in the micro controller hardware, having the Systick interrupt launch either every 100 μs (F4) or every ms (F0) and increment a timing counter. Because of how the "Systick" interrupts are implemented [34], increasing the resolution will mean that Systick generates interrupts more often, which in turn leads to higher overhead as more time will be spent on processing these interrupts. For example, if we want to have a 100 μs resolution, that means every 100 μs the device will receive an hardware interrupt, halting the current program. Then it will

process the interrupt (increasing our "time elapsed" counter) and return back to the spot in the program it left, a process which will take some clock cycles.

Performance penalty compared to running Systick at 1 ms resolution depends on the platform and the chosen Systick resolution, as shown in table below. Performance penalty for using 1 μs resolution for F0 was not measured, as it would have been unacceptably high in any case.

	Performance penalty	
Resolution	F0	F4
1 ms	**Starting point**	Starting Point
100 μs	0.9%	**Negligible**
10 μs	10.7%	1.8%
1 μs	N/A	22.5%

Table 4: Performance penalty associated with using Systick with different timing resolutions; timing resolution chosen for measurements on each platform is shown in bold

5.2 Cryptographic performance

The performance of the uECC library was evaluated by measuring the time required for each task in the following sequence: First, a key-pair was generated, then the private key from the pair was used to sign a hash, and the public key from the key-pair was used to verify the signature created in previous step. The final step was to compress and then decompress the public key created in the first step. The sequence was run 100 times and average values were calculated from the results, shown in Table 5.

Tests were run on both platforms. For F0, there are optimized mathematical libraries written in assembler available in uECC and thus tests were run twice on F0: once using assembler optimizations, once without them. No assembler optimizations were available for F4.

For all other tests, F0 was set to use assembler optimized libraries as they significantly increase the performance of F0.

For each signature the Barrier has to verify, it first needs to decompress the public key and then to verify it. As per table 5, F4 takes 7 ms to decompress and then 66 ms to verify, meaning F4 will spend 73 ms per signature doing cryptographic calculations. For F0, using assembler optimizations, these values increase to 52 ms and 450 ms respectively, and thus requiring 502 ms to verify each signature, almost a seven-fold increase in time taken.

To complete the transaction, the Barrier has to verify N+1 signatures, where N is the amount of certificates embedded in the service request, with N being minimum of 1. For our normal use case of 4 certificates this means 5 signatures and in the case using certificate chain reduction, 2 signatures.

Operation	F4 no assembler	F0 assembler	F0 no assembler
Key generation	59 ms	410 ms	630 ms
Sign	64 ms	430 ms	650 ms
Verify	66 ms	450 ms	690 ms
Decompression	7 ms	52 ms	81 ms
Generate and sign a receipt	65 ms	440 ms	650 ms

Table 5: Time taken by cryptographic operations using uECC

After the transaction is completed, the Barrier has to generate and sign a receipt for the Client, a process which takes approximately 65 ms on F4 and 440 ms on F0.

Time required to calculate hashes depends on the length of the message portion being parsed, as seen in the Table 6, with the certificate message portion taking longest time. In Table 6, hash calculation is only considered for the service request message portion, not for the complete service request with an embedded certificate chain.

5.3 Message parsing and evaluation

Time required to parse the message portions depends on the type of the message portion being parsed. As seen in Table 6, a certificate takes longer to parse than a service request message portion, because the certificate is longer and also contains more fields.

Operation	F4	F0
Parse a certificate	0.10 ms	0.9 ms
Calculate hash for a certificate	0.43 ms	2.9 ms
Parse a service request	0.09 ms	0.7 ms
Calculate hash of a service request	0.38 ms	2.6 ms
Evaluate certificate chain of 1 certificate	0.05 ms	0.24 ms
Evaluate certificate chain of 10 certificates	0.07 ms	N/A

Table 6: Parsing, hashing and evaluation performance

Time spent on evaluating the certificate chain depends on the chain length. For F4 it takes 0.05 ms to evaluate certificate chain length of 1 and 0.07 ms for chain of 10. Because F0 has limited memory available, it was not implemented to support certificate chain length of 10 and thus tests were only ran for 1 certificate, with F0 taking 0.24 ms to evaluate a single certificate.

5.4 Bluetooth Low Energy performance

The Bluetooth module is an independent module, so its performance can be tested completely separately from the chosen embedded platforms.

It was evaluated from two different perspectives: First, I was interested in how long it would take for the Client to find the Barrier and open communications with

Operation	Time required
From starting software to "paired" state	120 ms
From paired state to receiving first packet from Barrier	230 ms
From starting software to receiving first packet	350 ms

Table 7: Bluetooth LE connection operations and times required

it. The starting point for that is service discovery: how long does it take for the Client to receive Advertisement sent by the Barrier. The second step is opening the actual communications link and negotiating transmission rates etc.

As seen on Table 7, pairing/service discovery is quite a fast operation, taking approximately 120 ms, but opening the link and receiving the first message adds around 230 ms, so it takes 350 ms to reach the point where the Client can start sending the signed service request to the Barrier.

The second perspective was to evaluate the actual maximum transfer speeds, which was done for both scenarios in our use case: sending a signed service request with a certificate chain length of 4, or the Certificate Chain Reduction scenario where the certificate chain length is 1.

With a certificate chain length of 4, the time required to receive all 3716 bytes of payload was on average 870 ms, resulting in an effective throughput of approximately 34 kpbs.

For a signed service request with a certificate chain length of 1, time required to receive all 1442 bytes of payload was on average 330 ms, resulting in an effective throughput of approximately 35 kbps.

As stated in Section 4.3.4, delays had to be inserted to Bluetooth transmission speeds on F0 so that message buffers would not overrun, but these artificial delays do not represent the actual performance the BLE module is capable of, and therefore are not taken into account here. The effect on performance created by the delays is negligible, because the F0 based system is not bandwidth bound.

5.5 Transaction performance

Previously I have measured both the time required to transmission the message and the time required by the computational operations. However, transaction performance cannot be calculated in a linear manner from the figures above, as computational operations are pipelined to be partially completed during message transmission. So transactional performance consists of three pieces which were measured: the time elapsed between the Client sending a signed service request to the Barrier, and the Barrier having completed evaluating the certificate chain, time required for device discovery and finally the time required for opening a connection, negotiating link speed etc., with last two taking on average 350 ms.

In the Table 8 I have collated the performance measurements for all four system performance profiles, with two use cases, run on two platforms: Our standard use case with certificate chain length of 4, and Certificate Chain Reduction use case with certificate chain length of 1, both tested on F0 and F4 platforms.

Operation type for time spent	F4, 1 certificate	F4, 4 certificates	F0, 1 certificate	F0, 4 certificates
Total time taken to complete transaction	760 ms	1300 ms	1700 ms	3200 ms
Bluetooth				
Service discovery	120 ms	120 ms	120 ms	120 ms
Opening Connection	230 ms	230 ms	230 ms	230 ms
Time spent transferring information	330 ms	870 ms	330 ms	900 ms
Time spent delaying	N/A	N/A	30 ms	630 ms
Computation				
Parsing and hashing	3 ms	5 ms	16 ms	40 ms
Decompress and verify	150 ms	370 ms	1000 ms	2530 ms
Combined				
Bluetooth Only	610 ms	920 ms	610 ms	640 ms
Computational operations done during Bluetooth transmit and delays	75 ms	300 ms	98 ms	1300 ms
Computational operations only	74 ms	74 ms	930 ms	1300 ms
Signed service request				
Generate, sign and transmit	400 ms	400 ms	780 ms	780 ms

Table 8: Transaction performance

For F4, the transaction takes less than 1.5 seconds in the case with 4 certificates and less than 1 second using Certificate Chain Reduction. For F0, the performance is significantly worse, taking 1.7 seconds to complete even the shorter transaction with CRC applied and 3.2 seconds for the case with 4 certificates.

A transaction is considered complete, when the Barrier has made the decision about granting access to the parking lot. After the transaction is complete, the Barrier will generate, sign and send a signed receipt to the Client. This process is heavily bandwidth bound, taking approximately 400 ms for the F4 and 780 ms for the F0 because the current implementation is not optimized to pipeline signing to take place while the message is being sent.

Receipt generation and sending does not have impact on user experience, as the time required to open the gate and drive a car out of Bluetooth Low Energy range is significantly higher than 1 second.

6 Analysis

In this section, I will present analysis, discussing each Research Question presented in the Introduction (Section 1.3): Transaction performance (Research Question 2), Task distribution (Research Question 2), Relative cost of computational tasks (Research Question 3), Certificate Chain Reduction (Research Question 4) and Benefits of public key compression (Research Question 5).

I will also evaluate the performance of Bluetooth Low Energy and discuss the security aspects of the system.

6.1 Research Question 1: Transaction performance

Research Question 1 was: how long will it take to complete the transaction and how will our embedded platform choice affect it? As seen in the figure 14, using

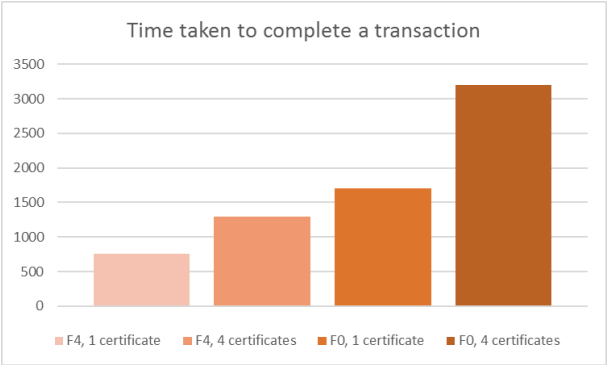


Figure 14: Transaction times by platform and certificate amount

F4 we were easily able to complete the whole transaction in less than 1.5 seconds even when using 4 certificates and in less than 1 second when taking advantage of Certificate Chain Reduction, so the system is definitely fast enough for our purposes as defined by our Business Requirement 6. Latency of less than one second should be considered good even for cases where there is human interaction and the process is not something repeated extremely often.

F0 almost meets our requirements when using Certificate Chain reduction, taking 1.7 seconds to complete the transaction. Unfortunately when using 4 certificates, the transaction takes 3.2 seconds, leading to a very noticeable delay between starting and completing transaction. I believe that using a certain optimizations which will be discussed later in Chapter 7, it would be possible to complete the whole transaction in slightly under 1.5 seconds.

In conclusion, I find F4 a very suitable embedded platform for this application. In cases with less strict time requirements, for example machine-to-machine interactions without strict time limits, F0 would be fast enough, though its limited memory might pose other challenges.

6.2 Research Question 2: Task distribution

In this section, I analyze the task distribution during the transaction. As the system performance profiles for F4 and F0 are different, they will be analyzed separately.

6.2.1 F4

Timeline diagrams, Figures 15 and 16, show the distribution and timing for each task on F4 platform (times to transfer each message portions are approximated). It is evident from the diagrams that majority of the time is spent on Bluetooth Low Energy activities. Purely computational operations that cannot be pipelined account for less than 10% of the time budget in both cases, as seen on Figure 17.

As seen on Figures 15 and 16, the last message portion to be transmitted is always the signature for the service request. Thus no matter how fast cryptographic operations are calculated, this signature must always be verified after the Client-to-Barrier Bluetooth transmission is complete.

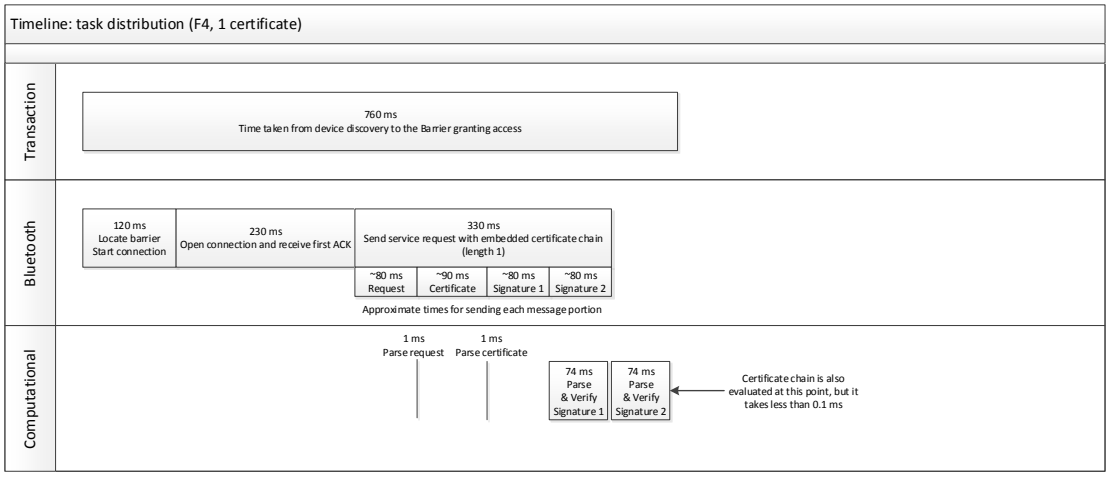


Figure 15: Times spent on tasks in transaction. F4, 1 certificate

Bluetooth dominates the time budget for two reasons: first, there is significant overhead (350 ms) in just establishing communications between Client and Barrier, where no payload except Nonce and Barrier ID are transferred. But even if that is discounted, the system still spends the majority of time just waiting for data to arrive via BLE: at 34–35 kbps, each message portion takes around 80–90 ms to transmit.

Considering that only half of the message portions are signatures that have to be verified, it means that the Barrier spends around 170 ms waiting for a message to arrive for each 74 ms the Barrier spends verifying the signatures: in other words, bandwidth is a major bottleneck in our performance for F4 platform.

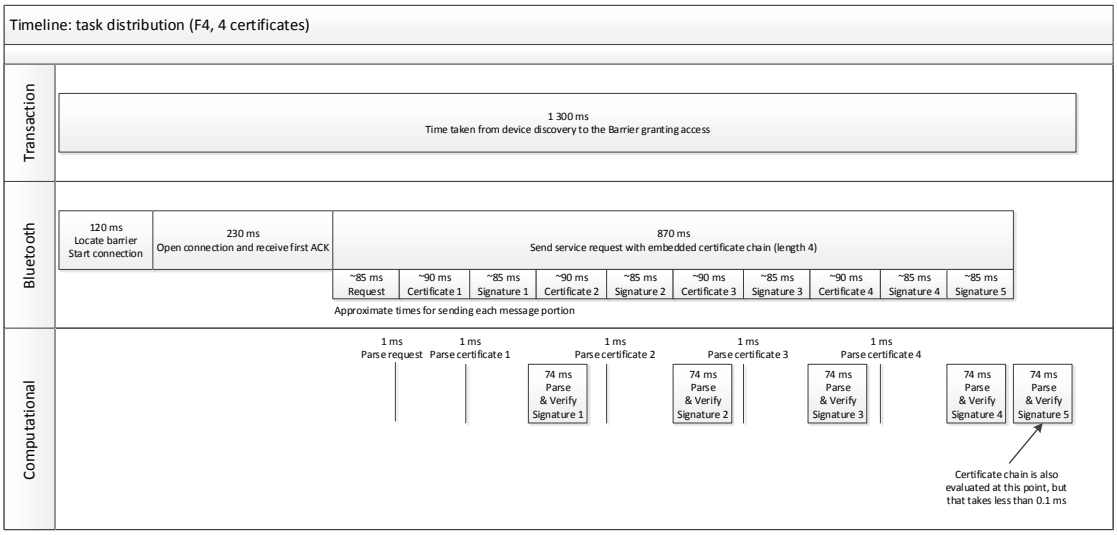
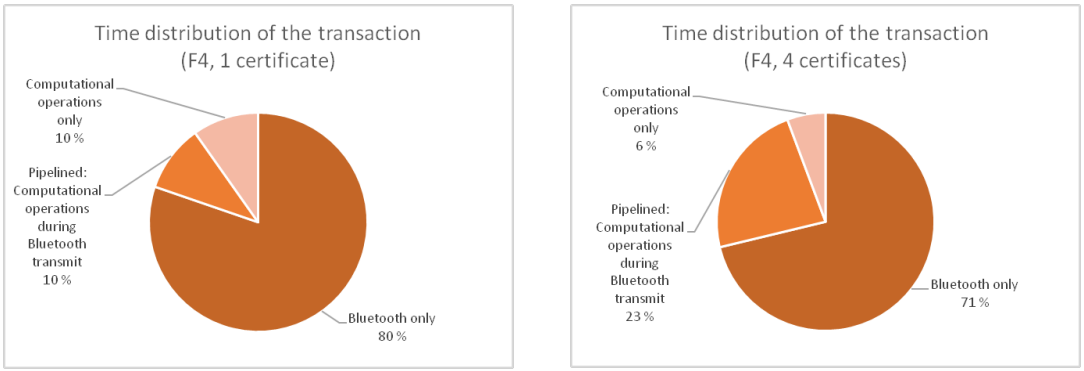


Figure 16: Times spent on tasks in transaction. F4, 4 certificates



(a) 1 certificate

(b) 4 certificates

Figure 17: Time distribution between Bluetooth and computational tasks for F4.

6.2.2 F0

For F0, situation changes radically from F4. This is easy to see in the timeline Figures 18 and 19, where it can be seen that F0 is unable to complete signature verification before receiving the next message. The large delays shown in Bluetooth Low Energy transfers in Figure 19 had to be added to keep the Barrier's message buffer from overflowing. Still, the inserted transfer delays only add 20 ms to the time taken by the whole transaction, as now the system bottleneck has moved from bandwidth to computational performance.

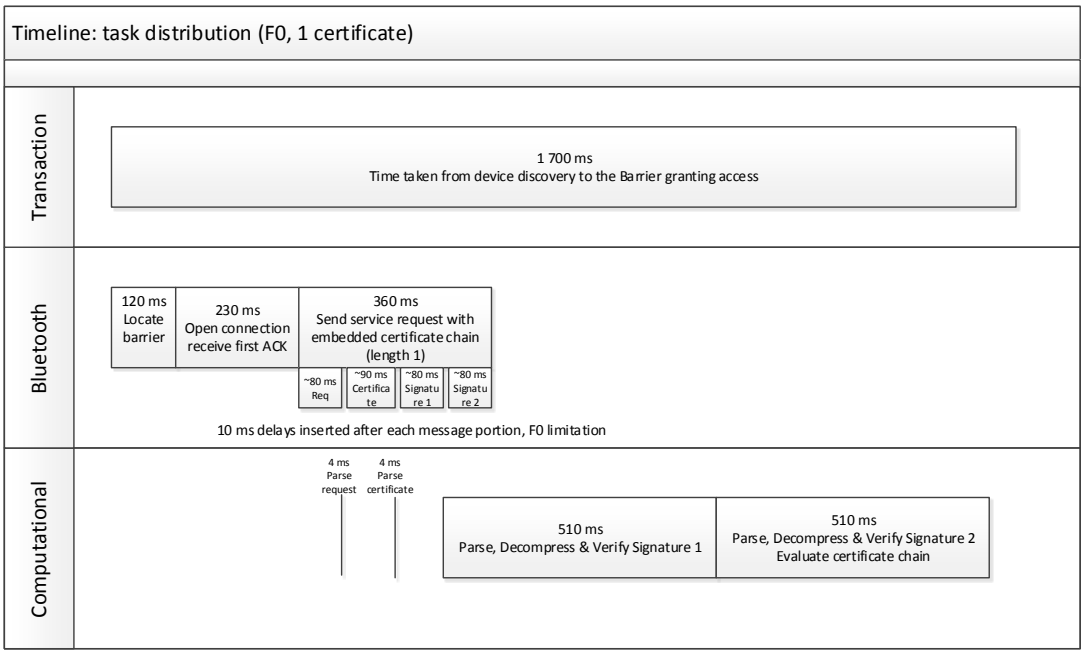


Figure 18: Times spent on tasks in transaction. F0, 1 certificate

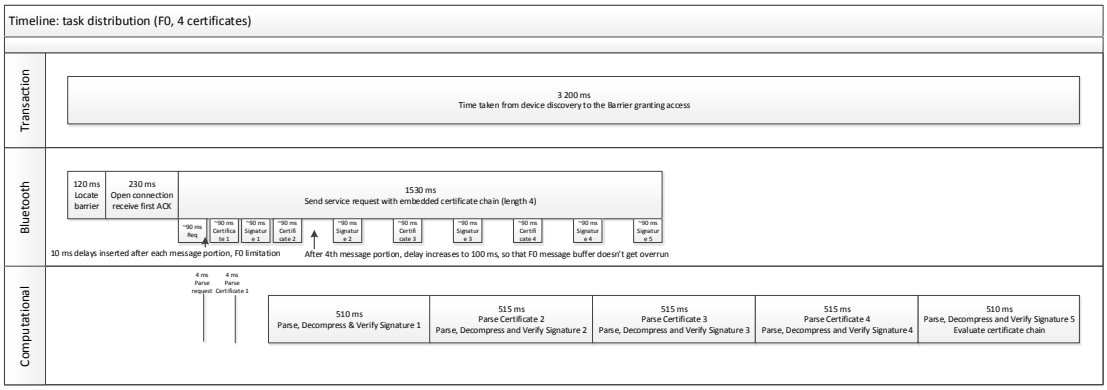
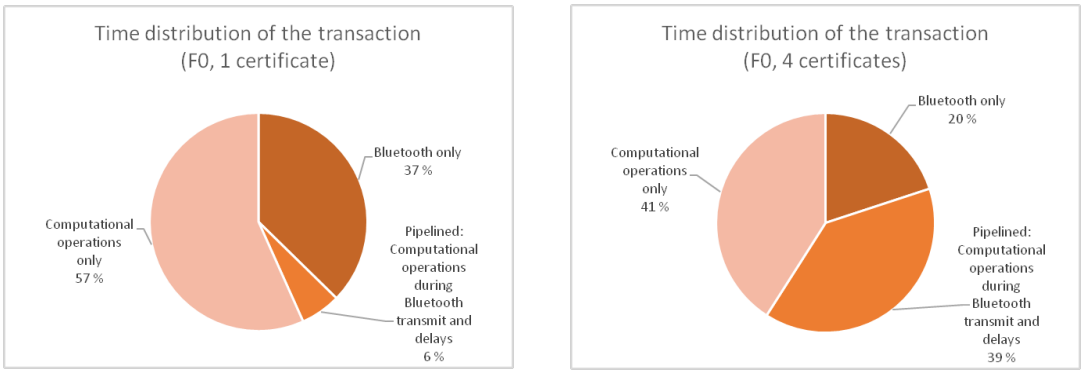


Figure 19: Times spent on tasks in transaction. F0, 4 certificates

As seen on the pie graphs in Figure 20, Bluetooth drops from dominating the time

use to being less than half for 1 certificate case. Pie chart b in Figure 20b is effected by the fact that the added delays move computational operations from "Computational operations only" to "Pipelined". If the delays were removed, "Computational operations only" would increase to 57% and the time taken for the transmission could be almost halved, but this would have only a minimal effect on the actual transaction performance.

For F0, the main bottleneck is computational performance, whereas for F4 it's bandwidth.



(a) 1 certificate

(b) 4 certificates

Figure 20: Time distribution between Bluetooth and computational tasks for F0.

6.2.3 Conclusion

In conclusion, while the actual tasks remain the same, distribution of used time is dependent on the processing power of the embedded platform. For F4, the nRF8001 BLE chip is an obvious bottleneck in the system performance and if increased performance is desired, increasing bandwidth is priority number one.

For slow embedded systems like F0, performance characteristics of the communication platform become less important, especially if Certificate Chain Reduction is not used, as cryptographic operations dominate the time budget.

6.3 Research Question 3: Computational tasks

The computational tasks the embedded system performs are parsing, evaluating certificate chain, hashing, decompressing public keys and verifying signatures, with last three being considered cryptographic operations in the scope of this work.

In most cases, time taken by these processes scales linearly with the number of times each task has to be done. There are small differences, for example parsing and hashing are dependent on the length of the message portion which means that those tasks take longer to perform for a certificate than they do for a service request or signature, but generally the time required for computational tasks scales on O(N), where N equals 1 + certificate chain length.

The major exception to this is certificate chain evaluation: considering that evaluating a certificate chain of length 10 takes less than twice the time required for evaluating a certificate chain of length 1, O(N) scaling doesn't seem appropriate for that operation. I did not find the reason for this, but the most likely explanation is some relatively high-cost operation that only needs to be done once per evaluation.

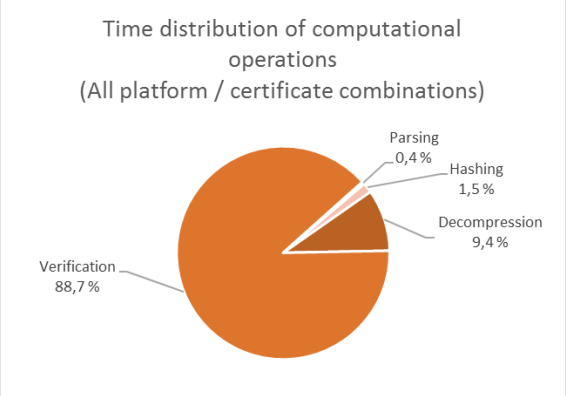


Figure 21: Computational time distribution for all platform/certificate combinations

I was interested in the time distribution of the different computational tasks. As seen on the Figure 21, from the performance viewpoint verification and decompression dominate computational operations. When one considers that hashing is required in this system for verification, of all computational operations those related to cryptography take over 99% of resources, with parsing and evaluation taking less than 0.5% of time.

So it can be said that from a computational performance viewpoint, only cryptographic operations matter.

6.4 Research Question 4: Certificate Chain Reduction

Figure 14 shows that using Certificate Chain Reduction gives large performance benefits: Not considering the time taken by service discovery and opening Bluetooth link, when using 4 to 1 Certificate Chain Reduction the time required to complete a transaction drops by 55% and 51% on F4 and F0 respectively. When taking the

whole transaction into account, the time reduction is reduced to 41% for F4 and 49% for F0, which is still a very significant performance gain.

The reason why the reduction is smaller for F0 than it is for F4 is that fixed time costs of opening a Bluetooth link (350 ms) are a relatively much greater portion of the transaction for F4 than they are for F0, 44% compared to 21%.

Chain Reduction doesn't just improve performance: it also allows embedded devices to save energy, as the energy usage of a properly designed embedded system is heavily dependent on the processor time required to perform tasks. Being able to halve the amount of computation required by the embedded system will lead to major energy savings. While energy savings are not as important in all contexts, for battery powered devices, energy usage often is what makes or breaks a system.

For the following reasons, reducing the amount of certificates via Certificate Chain Reduction does not increase performance in a linear fashion:

1. While the amount of verify operations required is a function of the number of certificates in the certificate chain, a signed service request is also always present. So the amount of operations required is N+1, where N is the amount of certificates present in certificate chain. 4 to 1 reduction then reduces the number of verifications required from 5 to 2, a 60% decrease.

2. As signature for the service request is always the final message portion received, so there is always one cryptographic operation that cannot be pipelined.

3. Certificate Chain Reduction does not reduce the amount of time required for establishing communications.

6.5 Research Question 5: Benefits of public key compression

There is a trade-off in using compressed public keys, between the required bandwidth and the time taken by decompressing required public keys. My hypothesis was that the time saved by the decreased message length would more than compensate for the extra effort required to decompress public key. This hypothesis was only partially correct, being true for F4 and false for F0.

Each service request and certificate message portion stores two public keys and each signature message portion stores one public key, leading to each signed message portion having three public keys. Because only the public key stored in the signature portion has to be decompressed by the Barrier, 186 bytes (3 public keys, each compressed public key being 31 bytes shorter than uncompressed one, using hexadecimal encoding) are saved by using compression.

This means that when using compressed public keys with current maximum bandwidth of 35 kbps, for each signature portion sent system gains 43 ms in saved transmission time and spends either 7 ms (F4) or 52 ms (F0) doing decompression. For F4, the using compressed public keys saves more than the 36 ms (43 ms - 7 ms) per signature sent, as the transaction is bandwidth bound, and for all pipelined operations system is able to complete both public key decompression and signature verification before next message portion has arrived (see Figures 15 and 16). This means that F4 gets the benefits for all signatures, but pays the "price" of decompression only once (for the final signature that cannot be pipelined) as far as performance is concerned.

Bandwidth (bps)	Time saved (ms)
236 700	6.3
230 400	6.5
115 200	12.9
57 600	25.8
33 400	44.6
28 800	51.7
14 400	103.3
9 600	155.0

Table 9: Time saved by transferring three public keys in compressed format by bandwidth

For F4, using compressed public keys saves approximately 79 ms of time (10%) for the Certificate Chain Reduction Scenario and 208 ms (16%) for the 4 certificate scenario.

For F0, situation changes. With almost seven-fold reduction in processing speed, using compressed certificates leads to slightly decreased transactional performance, increasing time taken by 18 ms (∼0.5%) when using Certificate Chain Reduction and 45 ms (∼1%) for the 4 certificate scenario. This performance penalty is marginal and doesn't mean using compressed public keys for F0 is a poor choice: F0 has only a limited amount of memory and storing public keys in uncompressed form would require allocating over 30% more memory for the data structure storing message portions, compared to using compressed public keys.

The breakpoint for using compressed public keys is calculated in Table 9: we would need to increase our current achieved throughput by at least factor of 6 to 210 kbps for uncompressed public keys to have a performance advantage over compressed ones (using F4).

Implementing a denser encoding would, of course, reduce time saved by using compressed keys: for straight binary encoding numbers in table 9 should be halved, leading to decreased absolute effectiveness. Denser encoding could, on other hand, also mean that the relative effect of public key compression is increased as a larger portion of the message would consist of public keys, because in current encoding large portion of the message consists of verbose field descriptions that could be replaced with short codes.

6.6 Bluetooth Low Energy performance

Though the theoretical maximum bandwidth of Bluetooth Low Energy is 236.7 kbps [12], the actual maximum is much lower: each packet has a maximum of 20 bytes of payload and a device could send or receive 6 such packets per minimum interval of 7.5 ms, leading to a maximum throughput of 128 kpbs. Unfortunately, no such device is currently available to function as the Client device: while Apple iOS devices support using 6 packets per interval, the minimum interval Apple devices can use for communicating with generic BLE devices is 20 ms [1] which leads to a maximum throughput of 48 kbps.

Android devices, on the other hand, do support the minimum 7.5 interval defined in Bluetooth specification [13]. Unfortunately they only support 4 packets per interval at that speed, which leads to a theoretical maximum throughput of 85 kbps, though in real life only speeds of 58 kbps have been achieved [12]. Unfortunately the chosen BLE chipset (nRF8001) only supports receiving 2 packets and sending one packet per interval at 7.5 ms, reducing the theoretical maximum throughput to 42.6 kbps. Testing shows that even said speed is not achievable, with 34-35 kbps being a realistic maximum for this combination.

While these numbers are heavily dependent on the software Bluetooth Stack used in the Client device, most likely other Android devices have similar performance. Apple iOS devices on the other hand will have a lower maximum throughput using nRF8001, 50% to 75% compared to Android.

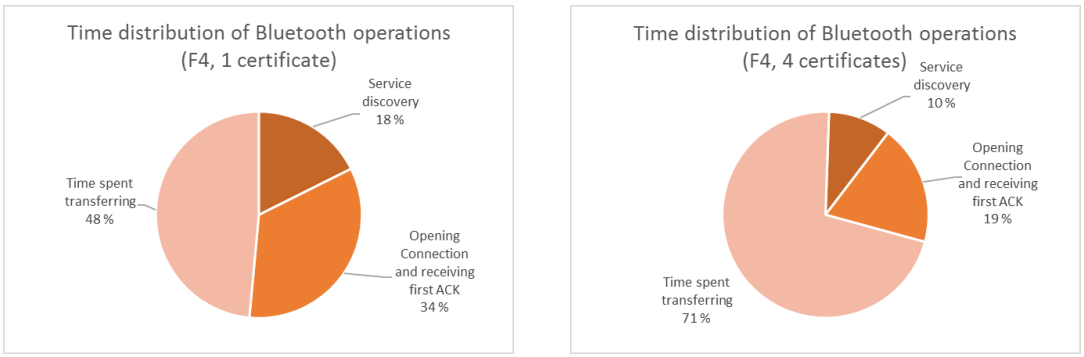
Bluetooth Low Energy suffers a small, but measurable, drop when sending larger payloads. The reason for this is unknown, but most likely related to the Android Bluetooth stack: I found a bug in it where a small (1 ms) delay had to be used before inserting next packet into BLE transfer queue or a non-deterministic packet loss of 1-2% would occur on longer transmissions. This added delay was not evident on the actual transmission side of the Bluetooth Stack though: the time taken to insert packets into BLE transfer queue was less than 1/3 of the time required to transmit the same BLE packets.

While our modular architecture should in theory be able to make both computational modules have same Bluetooth performance, the delays required by limitations in F0 platform do change the performance profiles and because of this, F4 and F0 are discussed separately.

6.6.1 F4

As seen in the Figure 22, the time distribution of Bluetooth operations for F4 is quite simple: a fixed time to required to open a connection accounts for 30-50% of the Bluetooth time budget. Changing to a BLE chip with higher bandwidth would not result in a linear performance increase, and a more than 2x increase in bandwidth would lead to diminishing returns as the system would become CPU bound.

Increasing the bandwidth to 2x would cut approximately 130 ms from the total transaction time when sending one certificate and 390 ms when sending four, but after that the point of diminishing returns is reached. Further increasing the



(a) 1 certificate (b) 4 certificates

Figure 22: Bluetooth time distribution for F4.

bandwidth to 4x would only result in savings of approximately 190 and 490 ms, respectively.

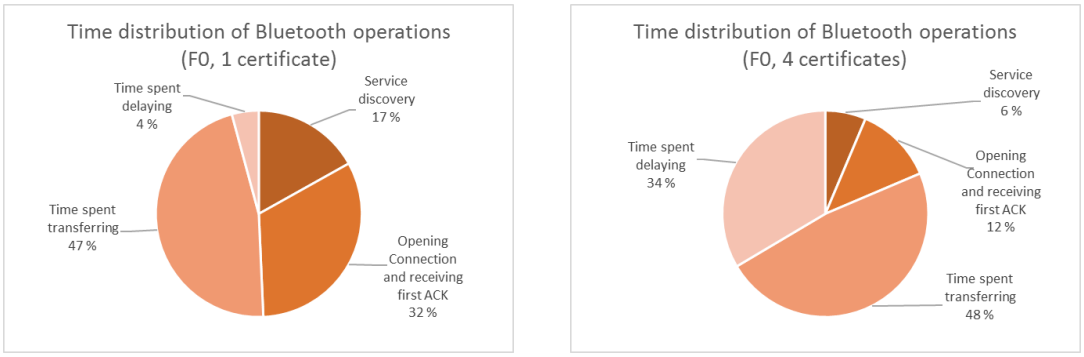
The reason for the non-linear savings increase is the message structure of a signed service request: the third message portion is the first to require cryptographic verification. Before the third message portion is received, the system can benefit from all the increased bandwidth, but after said message portion has been received, any bandwidth increases above approximately 2.2x will result in the system becoming CPU bound.

6.6.2 F0

Unfortunately, the F0 limitations discussed earlier meant that artificial delays had to be inserted to the Bluetooth Low Energy module. For the Certificate Chain Reduction use case, delays are pretty marginal as is seen in the Figure 23, as sending only 4 message portions means that only the last message portion has to be stored in the circular buffer while the previous message portion is still being processed, (see figure 18), but for the use case of 4 certificates, delays account for 34% of all the time taken for Bluetooth.

Fortunately, these only affect the time taken for actual transaction minimally as described in Section 4.3.4.

Interestingly, even though we are forced to add delays to Bluetooth transfers, increased bandwidth would still be useful from a performance point of view. While the performance increase would not be as great as for F4, because of the signed service request message structure explained in previous section, performance increase would still be almost as large for the use case with one certificate, as it is for F4.



(a) 1 certificate (b) 4 certificates

Figure 23: Bluetooth time distribution for F0.

6.7 Security

There are three common attack vectors present for any system that transmits information:

1. Passive Wiretapping, or eavesdropping, where an attacker can receive a copy of private communications, thus compromising the confidentiality of the information.

2. Man-in-The-Middle (MITM), where an attacker can insert or alter messages between the communicating parties, compromising both the confidentiality and integrity of the information.

3. Replay attacks, where an attacker eavesdrops secret information, and replays that information to gain access to the resource.

All three vectors have been taken into account when designing the protocol, even though Bluetooth Low Energy as a transmission channel is vulnerable to both eavesdropping and MITM-attacks[13] because session encryption keys are transmitted in cleartext. These attacks are mitigated by two ways:

All messages are digitally signed and thus it is not possible to insert, remove or modify any data sent between the Client and the Barrier without the other party noticing it and discarding the message. This makes most MITM attacks ineffective.

Replay attacks are prevented by using a nonce, which will make the Barrier deny all attempts to use the system by retransmitting eavesdropped message. Using a nonce means that eavesdropping becomes a privacy issue instead of a security one.

While the system is still technically vulnerable to possibly leaking private information, one must take into account that because Bluetooth Low Energy has such a short range, all attacks require physical presence at the system location. At this point, the attackers could probably get far more private information of the users simply by setting up a video camera and using Optical Character Recognition to read license plates.

It would also be possible to implement a separate layer of encryption, but the performance penalties would most likely be severe, and I do not consider the trade-off worth it in this case.

I chose not to sign Advertisement messages because it would significantly increase time required to complete a transaction, which could mean that spoofing a Barrier would be possible. While spoofing does not open any new evident attack vectors, in future it might be useful to give the Client the option to request a signature for the advertisement or to send it automatically. While using scan response packet would be ideal for this in all other respects, the problem is that scan response also has only 31 bytes of payload which is insufficient for any secure form of signature to be transmitted.

The newly released Bluetooth 4.2 specification has extensions for fixing the security issues present in Bluetooth Low Energy, but as of yet there are no devices implementing it.

As the current system doesn't support limits or online checks, it is vulnerable to certain types of misuses, unless they are addressed in some way. For example, a driver could use the same certificate to park multiple cars at the parking lot or a reseller might sell more parking places than there are free places available in the parking area.

Handling of these issues is just as much a business decision as a technical one: while it is possible, though extremely cumbersome, to add limits to the certificate issued by the reseller, it is much easier to make a contract that stipulates how many permits a reseller is allowed to sell.

Similarly, while it would easy to note which certificates are "in use" currently at each parking area, synchronize the information to a server and then have the Barrier check this before allowing a driver entry to the parking lot, from a business perspective it would probably be better to just log the certificate use to a central server that would check for concurrently used certificates once a day and send a bill for any misuses. This also has the additional advantage of not requiring a constant network connection.

F0 using a non-cryptographically secure pseudo-RNG to generate the random number k for purposes of signing does present a problem: as stated in Section 2.2.1, having non-random k leads to the possibility of the private key leaking. There are many possible answers to this, ranging from implementing a proper deterministic Random Number Generator as described in the document SP800-90A [3] to implementing true RNG using General Purpose Input/Outputs as entropy sources in the F0.

7 Discussion

The objective of this work was to find out if an access control system, using a smartphone as the Client device and an embedded device for verification, would be fast enough for parking purposes.

F4 embedded platform system is able to process a service request with 4 certificates in less than 1.5 seconds, and taking advantage of Certificate Chain Reduction to reduce number of required certificates to 1 speeds up the transaction so that it takes less than a second, exceeding our business requirements (Business Requirement 6).

To better understand the performance aspects, the system was also implemented with F0 platform, which almost met our performance requirements when using Certificate Chain Reduction, taking 1.7 seconds to complete transaction.

So while the goal was attainable, there are certain hardware requirements and cheapest microcontrollers will have difficulty fulfilling them. Another consideration is that cheaper embedded platforms rarely have hardware Random Number Generators, and ECDSA signing is susceptible to leaking the private key if either number gets re-used in multiple signings, or if the numbers can be guessed.

The advantage of using this system for access control, is that it requires relatively little existing infrastructure: it can be installed easily wherever a low voltage line can be installed. Being small, cheap and low on energy usage, it would be possible to install one on every parking spot at a garage, instead of just installing one at the entrance. Or installing one as an electronic lock in a normal door.

Another advantage is that it would be possible to give the end-users a right to delegate access to the parking lot: for example, companies could give out very short-term certificates for visitors. Same certificates could even be used to grant access to open doors to the building.

From a hardware point of view the prototype was cheap, with cost of less than 100 euros including both embedded platforms. By using a different BLE module and integrating it directly to the embedded device, it would be easy to lower the hardware costs below 30 euros. Further decreases might require custom circuit board design, for which the initial costs are so high that selling thousands, or tens of thousands, of units might be required to recoup them.

8 Future work

In this section, I will concentrate on F4, simply because F0 is so slow that using a separate hardware cryptographic processor would be required to accelerate its performance to acceptable levels if Certificate Chain Reduction is not used.

While our system has low energy usage if connected to mains, its energy budget is most likely too large for a battery powered system, though it would be useful to evaluate this aspect of the system in detail.

8.1 Bandwidth and transmission times

Bandwidth being a major bottleneck in our system, it is also the first place to look for performance improvements. There are two relatively easy changes that could be made to reduce transmission times: nRF8001, the BLE chipset used in the prototype, only supports sending two messages per interval. Changing to a chipset supporting four messages per interval could in theory double our throughput to 68-70 kbps, though research suggests that the realistic maximum is below 60 kbps [12].

Currently, certificates are encoded as canonical S-expression (with advanced transport extension), using only printable ASCII characters. This is very wasteful from a transmission point of view. For example, all numbers are encoded as printable hexadecimal numbers, meaning that the system has to send 64 bytes to transmit 32 bytes of information.

Considering this and the fact that the field descriptions can be over 20 bytes long, changing to binary or partially binary encoding would easily allow halving the message size. This would effectively double our perceived bandwidth (halving the time required to receive signed service request).

Implementing these changes could almost quadruple transmission performance, changing the bottleneck to be CPU again. Doubling perceived bandwidth should reduce transaction time on F4 to approximately 860 ms when receiving 4 certificates, and 590 ms when using Certificate Chain Reduction, reductions of 33% and 22% respectively. Quadrupling perceived bandwidth leads to diminishing returns, reducing transaction times on F4 to approximately 640 ms in the 4 certificate use case and to 510 ms when receiving 1 certificate.

According to our previous analysis in Table 9, it would still be beneficial to use compressed keys for F4 even if perceived bandwidth is quadrupled, but for slower microcontrollers it does mean that moving to uncompressed keys at that point would lead to performance improvements.

8.2 Embedded platform

Cryptographic performance could be increased in at least two different ways: firstly, uECC, the cryptographic library used, includes optimized mathematical function libraries written in assembler for Cortex M0 processors and using them effectively

increases the performance by half. Porting those libraries to work with Cortex M4 would probably give similar performance enhancements.

These assembler optimizations could also allow us to consider using cheaper and slower platforms for applications. With assembler optimizations it should be possible to use a cheaper Cortex M4 platform than our F4 and still achieve performance similar to F4. While the F0, 48 MHz Cortex M0, was almost acceptable for the purposes of this work, it could be adequate for many other tasks where slow latency does not detract from user experience.

Another way to use cheaper embedded platforms would be to offload Elliptic Curve Cryptography to dedicated hardware. This could reduce both the time and energy required to perform cryptographic operations by multiple magnitudes.

The limited amount of RAM in F0, or other cheap platforms, might pose a problem for applications built on that platform: I was already forced to reduce both the maximum allowed certificate chain length and circular buffer size to get it to work in our proof-of-concept program: it is possible that memory required for implementing a complete system might require dropping the circular buffer altogether, resulting in a significant performance penalty as pipelining is disabled. Memory requirements might even make implementation impossible for F0.

While storing certificates or message buffer in the RAM of the embedded device is not required, other options would most likely either result in a relatively large performance penalty or would be more expensive to implement than just using a more capable embedded platform.

8.3 Caching certificates

Another way to increase performance would be to cache commonly used certificates, by adding hashes of verified certificates to a data structure, and before verifying certificate, checking if the hash is present in the cache and skipping the signature verification step accordingly.

Storing the certificate validity field in the cache is not required (as the certificate will be evaluated as part of the chain), and would increase the size of the cache structure by large margin, but for caching purposes it would be useful to have a mechanism that would evaluate the validity of a certificate in cache. For example, the Barrier could do a validity check on a certificate before checking if said certificate is present in the cache. If the validity check for the certificate fails, the cache would be purged of that certificate. At this point the Barrier could immediately send the client a NACK, informing that a certificate is not valid anymore.

The main problem with the this caching approach is that it isn't really effective in increasing the performance of the system, unless the system is CPU bound and has ample memory. In cases where the system is bandwidth bound, like the current F4 implementation, there is no performance advantage for using this caching mechanism. It could be very useful for CPU bound systems like F0, but those usually lack the spare memory required for the caching mechanism to work, and adding external memory is very likely more expensive than changing to a faster microcontroller.

For bandwidth bound systems, a more complex caching system would have to be

implemented. One possible example would be a system where the Client presents hashes of the certificates it possesses to the Barrier, and the Barrier replies which of those certificates it doesn't require signature for, and having Client only transmit the certificates without signatures. Of course, if the Barrier would implement complete caching of certificates, the process could be made even faster.

When considering energy usage, a caching mechanism gives significant benefits. With the power usage of the embedded portion correlating almost one-to-one with processor time, any mechanism that will allow the system to sleep instead of churning processor cycles on expensive cryptographic operations will save a significant amount of energy. Even if the cache is large, accessing it is still basically free compared to the price of doing verification.

How effective the cache is depends on what environment the system is installed on. A good assumption for parking, especially if the parking places are rented on a monthly basis, is that most users will be regulars, meaning the certificates they use will only be verified once against the signature and then multiple times from the cache. Similarly, the number of users will most likely not be huge, which is good as the system can only store a limited amount of certificate hashes in the cache, though for F4 "limited" could mean a few thousands.

In summary, when the goal is to save energy, caching is always useful, but to gain performance improvements, the transmission protocol needs to take caching into account. Internal reductions are effectively a caching mechanism in all but name, and the main disadvantage of caching certificates is that it requires enough memory to store cached data for a large portion of the users.

8.4 Logging and network

For a commercial system, some sort of network connection would be desirable in future. It would be possible to create a system where a smartphone Internet connection is used for transferring data: having the Barrier transfer payload data to the Client and then having the Client transfer it to a server via Internet the next time it has an Internet connection. This is of course a quite complex setup with many possible problems, but it would enable creating a Barrier entirely without a wired network.

9 Summary

This thesis presents a prototype implementation of an SPKI Authorization certificate based access control system that would support distributed non-networked systems. My main goal was to first explore the feasibility of the solution and then, having found it feasible, evaluate the performance of such system.

The faster embedded platform was surprisingly fast, being able to verify a 256-bit Elliptic Curve cryptographic signature in less than 70 ms. This speed enables our system to perform a full access control transaction in less than one second using Certificate Chain Reduction and in under 1.5 seconds without, which is excellent performance for our parking application.

The Certificate Chain Reduction mechanism presented in SPKI offers major benefits, increasing the effective performance of the system by 70 to 95 percent. One large advantage the Certificate Chain Reduction offers is that it works equally well for both CPU and bandwidth bound systems, as it reduces resource usage from both in similar amounts.

Public key compression was found to give a significant benefit when the system is bandwidth bound, but for F0, which is CPU bound, it gave a marginal penalty. Using compressed public keys also requires 25% less memory to store certificates in the internal data structures, allowing longer chains to be used.

Bluetooth Low Energy was found to be more than adequate for the purposes of this system. While the device discovery in Bluetooth Low Energy was very fast, Bluetooth Low Energy throughput was found to be surprisingly low. This was partially because of the chosen Bluetooth Low Energy chip (nRF8001,) but the faster embedded platform would have been bandwidth bound even using a faster Bluetooth Low Energy chip.

While Bluetooth Low Energy communications are vulnerable to both eavesdropping and Man-In-The-Middle attacks, these are not security issues in this system, as all relevant communications are cryptographically signed and replay attacks are prevented via the use of a nonce.

The system was implemented without network connection and the delegation makes it possible to generate new certificates for the system without the original issuer of the certificate. For logging purposes, intermittent network connection would be useful, but it would be possible to transfer logging information using the Client device.

Our Proof-of-Concept system was cheap, the hardware required for the implementation cost less than 100 euros (excluding the Client device). At the time of writing, it would be possible to implement it with hardware costing less than 30 euros.

References

[1] Apple Inc. Bluetooth Accessory Design Guidelines for Apple Products, 2011. URL: `https://developer.apple.com/hardwaredrivers/bluetoothdesignguidelines.pdf` [Retrieved 2014-09-22].

[2] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for Key Management - Part 1: general (revision 3)(2012). *NIST Special Publication 800-57*, 800(July):1–147, 2012.

[3] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. *NIST Special Publication 800-90A*, 800(January), 2012.

[4] JW Bos, JA Halderman, and Nadia Heninger. Elliptic Curve Cryptography in Practice. In *Financial Cryptography and Data Security*, pages 157–175. Springer Berlin Heidelberg, 2014.

[5] BB Brumley and Nicola Tuveri. Remote timing attacks are still practical. *Computer Security-ESORICS 2011*, 216499, 2011.

[6] Matthew Burnside, Dwaine Clarke, Srinivas Devadas, and Ronald Rivest. Distributed spki/sdsi-based security for networks of devices, 2002. URL: `http://www.cs.columbia.edu/~mb/papers/burnside02spkisdsi.pdf` [Retrieved 2015-01-05].

[7] Certicom Research. Standards For Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography, 2009.

[8] Certicom Research. Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters, 2010.

[9] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. Spki certificate theory. RFC 2693, RFC Editor, September 1999. URL: `http://www.rfc-editor.org/rfc/rfc2693.txt` [Retrieved 2014-10-30].

[10] Simon Foley and Hongbin Zhou. Authorisation Subterfuge by Delegation in Decentralised Networks. In Bruce Christianson, Bruno Crispo, JamesA. Malcolm, and Michael Roe, editors, *Security Protocols SE - 13*, volume 4631 of *Lecture Notes in Computer Science*, pages 103–111. Springer Berlin Heidelberg, 2007.

[11] Helsinki Institute for Information Technology. SPIRE (Smart Parking for Intelligent Real-Estate). URL: `http://www.hiit.fi/spire` [Retrieved 2014-12-19].

[12] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, 12(12):11734–11753, August 2012.

[13] Bluetooth Special Interest Group. Bluetooth 4.1 Core Specification, 2013.

[14] Bluetooth Special Interest Group. Bluetooth 4.2 Core Specification, 2014.

[15] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, and Hans Eberle. Speeding up Secure Web Transactions Using Elliptic Curve Cryptography. In *NDSS'04*, 2004.

[16] Tuomas Ilola. *Indoor signal strength based tracking using Bluetooth 4.0.* Master's, Aalto University, 2012.

[17] Andrey Jivsov. Compact representation of an elliptic curve point. Internet-Draft draft-jivsov-ecc-compact-05, IETF Secretariat, March 2014. URL: `http://www.ietf.org/internet-drafts/draft-jivsov-ecc-compact-05.txt` [Retrieved 2014-11-25].

[18] Kmackay. micro-ecc github page. URL: `https://github.com/kmackay/micro-ecc` [Retrieved 2014-09-19].

[19] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203, January 1987.

[20] Yki Kortesniemi. *Managing the Usage of Authorisation Certificates.* Licenciate's thesis, Helsinki University of Technology, 2003.

[21] Yki Kortesniemi, Tero Hasu, and Jonna Särs. A Revocation, Validation and Authentication Protocol for SPKI Based Delegation Systems. In *Proc. Internet Soc. Symp. on Network and Distributed Syst. Security (NDSS)*, 2000.

[22] Yki Kortesniemi, Timo Kiravuo, Mikko Särelä, and Hannu Kari. Chain Reduction of Authorisation Certificates. *International Journal of Security and Networks*, in press, 2015.

[23] Yki Kortesniemi and Mikko Särelä. Survey of Certificate Usage in Distributed Access Control. *Computers & Security*, 44(July):16–32, 2014.

[24] Javier Lopez, Isaac Agudo, and Jose a. Montenegro. On the deployment of a real scalable delegation service. *Information Security Technical Report*, 12(3):139–146, January 2007.

[25] Peter Marwedel. *Embedded system design*, volume 1. Springer, 2006.

[26] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[27] VS Miller. Use of elliptic curves in cryptography. *Advances in Cryptology-CRYPTO'85 Proceedings*, pages 417–426, 1986.

[28] NIST. NIST.gov - Computer Security Division - Computer Security Resource Center. URL: `http://csrc.nist.gov/groups/ST/hash/policy.html` [Retrieved 2014-09-19].

[29] Juha Paajarvi. Xml encoding of spki certificates. Internet-Draft draft-paajarvi-xml-spki-cert-00, IETF Secretariat, March 2000. URL: `https://tools.ietf.org/html/draft-paajarvi-xml-spki-cert-00` [Retrieved 2014-12-10].

[30] R. L. Rivest, a. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[31] Ronald Rivest, William Frantz, Carl Ellison, Brian Thomas, Tatu Ylonen, and Butler Lampson. Simple Public Key Certificate. Internet-Draft draft-ietf-spki-cert-structure-05, IETF Secretariat, March 1998. URL: `http://tools.ietf.org/html/draft-ietf-spki-cert-structure-05` [Retrieved 2014-10-25].

[32] T. Saito, K. Umesawa, and H.G. Okuno. Privacy-enhanced access control by SPKI and its application to Web server. *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, pages 201–206, 2000.

[33] ST Microelectronics. STMicroelectronics - Microntrollers. URL: `http://www.st.com/web/en/catalog/mmc/FM141` [Retrieved 2014-11-30].

[34] ST Microelectronics. STM32F405xx, 2013. URL: `http://www.st.com/web/en/resource/technical/document/datasheet/DM00037051.pdf` [Retrieved 19.09.2014].

[35] Lee Thomason. TinyXML-2. URL: `http://www.grinninglizard.com/tinyxml2docs/index.html` [Retrieved 2015-01-06].

[36] Kevin Townsend. Introduction to Bluetooth Low Energy. URL: `https://learn.adafruit.com/introduction-to-bluetooth-low-energy?view=all` [Retrieved 2014-12-15].

[37] Virtual Application and Implementation Research Lab. Measurements of public-key signature systems, indexed by machine. URL: `http://bench.cr.yp.to/results-sign.html` [Retrieved 2014-12-29].

[38] H Wang, S Jha, T Reps, S Schwoon, and S Stubblebine. *Reducing the Dependence of SPKI/SDSI on PKI*. Springer, 2006.

[39] Erik Welsh, Patrick Murphy, and JP Frantz. Improving connection times for Bluetooth devices in mobile environments. In *Proc. Int'l Conf. Fundamentals of Electronics Communications and Computer Sciences (ICFS 2002)*, 2002.

A Service request and authorization certificate specifications for the parking use case

This appendix presents the specification for the service requests, authorization certificates and signatures for the use case of parking.

For each type, there is an example presented on how they can be broken down into their component S-expressions, and how those can then be further broken into subcomponent S-expressions.

Notations: If an S-expression is optional, is is marked by either ? or *: ? means 0-1, * means 0 or more. All numbers inside # tags are in hexadecimal notation

Certificate A Certificate object consists of the following S-expressions, coded in the canonical form: issuer, subject, delegation right (=propagate), access rights (=tag) and validity. A signed certificate consists of <cert> followed by <signature>

```
<cert>  = <issuer><subject><delegation><tag><validity>
```

Below is shown how the certificate object can be broken down into its components.

```
<cert> = (11:certificate<issuer><subject><delegation>?<tag><validity>)
  <issuer> = (6:issuer<principal>)
    <principal>  =  <pub-key> = (10:public-key<pub-sig-alg-id><key>)
      <pub-sig-alg-id> = (23:ecdsa-secp256k1-sha-256)
      <key> = (#<public key value in compressed form>#)
  <subject> = (7:subject<principal>)
    <principal>  =  <pub-key> = (10:public-key<pub-sig-alg-id><key>)
      <pub-sig-alg-id> = (23:ecdsa-secp256k1-sha-256)
      <key> = (#<public key value in compressed form>#)
  <delegation> = (9:propagate)
    <delegation> field is optional and if not present, certificate does not allow delegation
  <tag> = (3:tag<tag-expr>)
    <tag-expr> is completely domain specific: in our use case, it is following:
    <tag-expr> = (4:park<permit>*)
      Note:there can be multiple permits
      <permit> = (5:permit<ID><permission>)
      <ID> = (#<parking lot ID>#)
        8 byte ID of the parking lot in hexadeximal (32-bit)
```

<permission> = (#<permission type>#)
Permission type is coded as 2-byte bitfield, with first byte specifying permit value for weekdays and second for weekends. Setting bit 1 allows parking at the specified time, setting bit 0 disallows it

bit	7	6	5	4	3	2	1	0
time	0-6	6-7	7-8	8-9	9-16	16-17	17-18	18-24

<validity> = (5:valid<not-before>?<not-after><online-test>?)
<not-before> = (10:not-before<date>)
<date> = (19:YYYY-MM-DD_hh:mm:ss)
Optional
<not-after> = (9:not-after<date>)
<date> = (19:YYYY-MM-DD_hh:mm:ss)
Mandatory in this implementation
<online-test>
Online test consists of URL and public key
It is not used in this implementation

Service request
A signed Service request consist of the following S-expression: <service request> followed by <signature>. A service request consists of the following S-expression:

<service request> = <request><issuer><nonce><validity><auth-chain>

Which can be broken down further:

<service request> = (15:service-request<request><issuer><nonce><validity><auth-ch
<request> = (9:parking-at<ID><principal>)
<ID> = (4:area(#<parking lot ID>#))
8 byte ID of the parking lot in hexadeximal (32-bit)
<principal> = <pub-key> = (10:public-key<pub-sig-alg-id><key>)
<pub-sig-alg-id> = (23:ecdsa-secp256k1-sha-256)
<key> = (#<public key value in compressed form>#)
<issuer> = (6:issuer<principal>)
<principal> = <pub-key> = (10:public-key<pub-sig-alg-id><key>)
<pub-sig-alg-id> = (23:ecdsa-secp256k1-sha-256)
<key> = (#<public key value in compressed form>#)
<nonce> = (5:nonce<nonce-value>)
<nonce-value> = (#<hexadecimal value for>#)
Nonce value proving that this is correctly formed request
<validity> = (5:valid<not-after>?)
<not-after> = (9:not-after<date>)
<date> = (19:YYYY-MM-DD_hh:mm:ss)

Optional as nonce should be sufficient protection, but should be present

`<auth-chain> = (10:auth-chain<length><signed cert>*)`

<auth-chain> is also a S-expression specific to this implementation

`<length> = (#`<length of chain>`#)`

hexadecimal, 2 characters (0-255)

Minimum of 1, maximum length is implementation specific

Tells how many signed cert nodes there are in the auth chain

`<signed cert> = <certificate><signature>`

Mandatory: at least one, no more than <length>

Signature Certificate,service request and receipt also needs to be signed, but signature is not part of those message portions themselves. Signature consists of the following S-expression:

```
<signature> = <hash><principal><sig-val>

<signature> = (9:signature<hash><principal><sig-val>)
  <hash> =(4:hash<hash-alg-name><hash-value>)
    <hash-alg-name> = (7:SHA-256)
    <hash-value> = (#<SHA-256 hash in hex>#)
      Hash is calculated from the certificate/service request object (defined
      previously) and is 64 bytes long in string/hexadecimal form
  <principal> = <pub-key> = (10:public-key<pub-sig-alg-id><key>)
    <pub-sig-alg-id> = (23:ecdsa-secp256k1-sha-256)
    <key> = (#<public key value in compressed form>#)
  <sig-val> = (3:sig<sig>)
    <sig> = (#<signed hash>#)
      The signed hash is the actual signature calculated using ECDSA
```

Receipt A signed receipt consist of the following S-expression: <receipt> followed by <signature>.

```
<receipt> = <request reply><reason>

<receipt> = (7:receipt<request reply><reason>)
  <request reply> = (13:request-reply<hash-value>)
    <hash-value> = (#<SHA-256 hash in hex>#)
      Hash is calculated from the service request object (defined
      previously) and is 64 bytes long in string/hexadecimal form
    <reason> = (6:reason(#<reason code in hex>#))
      Reason codes are defined in the paper "A Revocation, Validation and
      Authentication Protocol for SPKI Based Delegation Systems"[21].
```

B Example of a signed certificate

This appendix presents an example of a certificate used in our use case. The certificate has been formatted so that it is easier for readers to parse: actual certificates do not have white spaces in them.

This certificate gives the subject the right to park at parking area 02, on weekdays between 00.00 - 18.00 (fe) and on weekends between 00.00- 17.00 and 18.00-24.00 (fd) and the certificate expires on the end of year 2015.

```
(11:certificate
    (6:issuer
        (10:public-key
            (23:ecdsa-secp256k1-sha-256)
            (#032db1ae60c8953545dcf404160c82723
            ce94b121d3c377e64d0abea87766437d9#)
        )
    )
    (7:subject
        (10:public-key
            (23:ecdsa-secp256k1-sha-256)
            (#03531e6c2be3ea0ff135b38ba9fcd0616
            d8affce93be131bda568b1d18e286b95d#)
        )
    )
    (9:propagate)
    (3:tag
        (4:park
            (5:permit
                (#00000002#)
                (#fefd#)
            )
        )
    )
    (5:valid
        (10:not-before
            (19:1970-01-01_00:00:01)
        )
        (9:not-after
            (19:2015-12-31_23:59:59)
        )
    )
)
```

The signed certificate continues on the next page.

```
(9:signature
    (4:hash
        (7:SHA-256)
        (#a99dcc8bba4062e937f2cb716253db091
        a113c39ffa9e2ab8759a3f85df41698#)
    )
    (10:public-key
        (23:ecdsa-secp256k1-sha-256)
        (#032db1ae60c8953545dcf404160c82723
        ce94b121d3c377e64d0abea87766437d9#)
    )
    (3:sig
        (#64272723bbf1c38e48af3456dd5491d0e
        192f278ac0e7aad485bebd1ae5a9ee9230f
        aa61cb08131f54610d2ae08a79019c3d9c7
        4944ab342015853a15351ca9d#)
    )
)
```